2019 20th IEEE International Conference on Mobile Data Management (MDM)

KOLQ in a Road Network

Zitong Chen*, Yubao Liu†, Ada Wai-Chee Fu*, Raymond Chi-Wing Wong‡, Genan Dai†
* Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong, China
† School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China
‡ Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology, Hong Kong, China
Email: ztchen@cse.cuhk.edu.hk, liuyubao@mail.sysu.edu.cn, adafu@cse.cuhk.edu.hk,
raywong@cse.ust.hk, daign@mail2.sysu.edu.cn

***Abstract*—Optimal location querying (OLQ) in road networks is important for various applications. Existing work assumes no labels for servers and that a client only visits the nearest server. These assumptions are not realistic and it renders the existing work not useful in many cases. In this paper, we introduce the KOLQ problem which considers the k nearest servers of clients and labeled servers. We also proposed algorithms for the problem. Extensive experiments on the real road networks illustrate the efficiency of our proposed solutions.**

***Index Terms*—KNN, Optimal Location Query, Road Network**

I. Introduction

Optimal location query is a basic operation in applications such as location planning, location-based service, location-based analysis on road networks[1], and profile-based marketing [1]–[11]. Consider a road network on which a set of clients and a set of servers are located, the goal of optimal location query (OLQ) in road networks is to obtain a location to build a new server, so that a certain objective function calculated based on the clients and servers is optimal, after the server is established at this location. The *MaxSum* objective [5] [7] [10] [11] is a function for an optimal location at which a new server can attract the greatest number of clients.

In existing work on OLQ, no class labels are given to servers. In reality, *class labels* such as the fast food brands KFC and McDonald's are typically associated with the servers. Servers with the same class label have a collaborative relationship, but servers with different class labels have a competitive relationship. When a new server with a class label is built, we want to maximize the benefit of all collaborative servers (which includes the new server) with the same class label, and the benefit is measured by the expected number of attracted clients.

The second limitation of existing OLQ algorithms is that clients are assumed to visit the servers nearest to them. In reality, clients may choose servers that are not too far. In this work, we allow a client to have a tolerance range defined by a number k for selecting servers. A client may visit any of its k nearest servers with a certain probability, and such probabilities for a client sum up to 1. The probability of visiting the i-th nearest server can vary from clients to clients. The assumption of previous work that a client visits only its closest server is a special case where the probability of visiting the nearest server is 1, and that for the next $k-1$ nearest neighbors is 0. It is noticed that the visiting probabilities of k nearest servers is also studied in the MaxBRkNN problem [12].

With the consideration of class labels and the visiting probabilities, we introduce a new objective called *KMaxSum* which denotes the expected weighted sum of the clients attracted by *all* servers with the same class label. The problem of KNN-based optimal location querying (**KOLQ**) in road networks with the KMaxSum objective is called *KMaxSum querying*.

To the best of our knowledge, this is the first study considering both server labels and KNN servers in road networks. The contributions of this paper can be summarized as follows. (1) We propose the problem of KOLQ with the KMaxSum objective, which is more natural compared with the existing OLQ. (2) An algorithm called *MAS* is designed for the KMaxSum querying, in which pruning techniques are introduced based on the idea of the k nearest location component (*KNLC*). (3) We conducted extensive experiments on the real world road networks of San Francisco (*SF*) and Colorado (*COL*) to show the efficiency of our proposed algorithms.

The rest of this paper is organized as follows. Section II reviews the related work and point out the difficulties of adopting existing OLQ methods to KOLQ. Section III gives the problem definition. Section IV introduce our proposed query algorithms. Section V reports on the empirical study and Section VI concludes the paper.

II. Related Work

Various topics on road network are studied in recent years. Some works focus on the efficiency of the shortest paths calculation, [13] proposes a well-separated pair decomposition method, based on which a path oracle is built to help retrieve an intermediated link in the shortest path. [14] proposes an efficient index, called distance signature, for distance computation and query processing over long distances. Some works focus on querying on the network, for example, the processing of KNN, continuous KNN queries, and reverse KNN on spatial network [15]–[17]. Among all kinds of querying, the optimal meeting point (OMP) query [18], is also a location

[1]http://www.esri.com/software/arcgis/extensions/networkanalyst

2375-0324/19/$31.00 ©2019 IEEE
DOI 10.1109/MDM.2019.00-71

determination problem, but with different target to ours. Its goal is to find a location that minimizes the sum of network distance from a given set of points. One important discover of OMP is the location should be one of the split points. For a point p in the given point set, a point x on edge (u, v) is called a split point if the shortest path from p to x going through u has the same length as the one going through v. However, this discover can not be applied in our problem.

The optimal location query, originating from the facility location problem, also known as location analysis [1]–[4], has been extensively studied. Recently, researchers in the database community are paying attention to this problem because of its broad applications, especially for road networks. We highlight some of the related work below.

The goal of MaxBRNN problem [1] is to obtain an optimal area to establish a new server to attract the maximum clients. The first polynomial-time complexity algorithm for the problem was introduced in [19] and some extensions of this algorithm were studied in [20]. An approximation approach was introduced for the MaxBRNN problem in [21]. An improved algorithm for MaxBRNN was given in [22].

Zhou et al. [12] first pointed out that clients may have different probabilities visiting their k nearest servers, and MaxBRkNN, a generalization of MaxBRNN, in the L_p-norm space was studied. A region partition based algorithm, MaxFirst, was proposed to solve MaxBRkNN effectively. However, this work is very different from ours. We explain as follows. Firstly, the objective functions are different since we consider class labels of servers. In KOLQ, clients attracted by the servers similar to the new server (i.e. all servers with the query label) are considered. With the query label, the relationship between the clients and the servers becomes more complicated. For example, supposed $k = 2$, a client has equal probability to visit his nearest server and second nearest server, i.e. $\widetilde{P}_1 = \widetilde{P}_2 = 0.5$, and there is only one client with KFC as his nearest server and McDonald's as second. The solution for MaxBRkNN is to make the new server the nearest server. Consider two queries of KOLQ. Query 1: The query label is KFC. In this case, whether the new KFC becomes the nearest server or the second, KFC will attract the client totally. Query 2: The query label is McDonald's. Wherever the new McDonald's is, McDonald's cannot totally attract the client, hence the best solution is anywhere. We can see that the solutions of KOLQ are totally different from MaxBRkNN. Secondly, our KOLQ problem is based on the road network environment instead of the L_p-norm space. MaxFirst benefits from the easy judgement of the intersection between a rectangle partition and an NLC (Nearest Location Component), which is a circle in L_p-norm space. However, similar judgement will be complex in road network since an NLC becomes a set of line segments. Another problem for adapting MaxFirst to road networks is that the lower bound of a rectangle partition is not achievable since an NLC in road network cannot cover a rectangle partition.

The OLQ problem with road network setting was firstly studied by Xiao et al. [5], and an efficient algorithm was presented. An extension of the OLQ problem with dynamic clients and servers was studied in [7]. Recently, a more efficient algorithm for the OLQ problem was proposed in [10], [11]. [23] first studied the exact solution for the OLQ problem with multiple new servers. [24] studied a static version and a dynamic version of OLQ with the MaxSum objective. There is a study about isochrone queries in a multimodal network [25], which is similar to the OLQ query. Xu et al. [26] studied the problem of proximity queries among sets of moving objects in road networks. The OLQ without the customer locations was studied in [27].

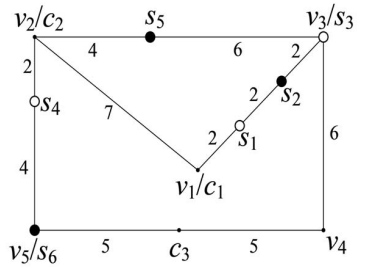
The existing OLQ does not consider server labels. Even if OLQ considers server labels, it is still different from KOLQ. First we explain how OLQ may handle labels. Given a query server label l, our target is to build a new server with the query label l to maximize the number of clients attracted by all the servers with label l. Then we can simply remove those clients which have already been attracted by the existing servers with the query label. After that, we build a new server that can attract the most remaining clients, which becomes the problem of OLQ without label. However, in KOLQ, clients are attracted to servers with probabilities, which may change after the new server is built, therefore, we cannot easily transform KOLQ with server labels to OLQ without server labels.

The algorithm in [28] was proposed to find an optimal location instead of an optimal region for the L_1-norm space. The algorithm in [29] was to obtain a location to build up a new server so that the average distance from each client to its nearest server is minimized. The problem studied in [8] [9] was to choose a location from a given set of potential locations to build up a new server, in order to minimize the average distance between the client and its nearest server. Compared to the aggregate nearest neighbor queries [30]–[32], we try to find an optimal location for a new object instead of a set of given objects. Though the objectives for the BRNN problem [33] and the reverse top-k problem [34] are similar to ours in some way, their algorithms cannot handle road network. The probabilistic reverse nearest neighbor queries studied in [35] [36] is also related to BRNN but they consider uncertain databases.

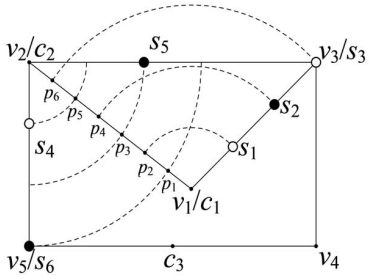
Our discussion in the above shows that KOLQ cannot be solved by existing algorithms for OLQ or BRkNN. We need to design new algorithms for this problem. The candidate points for the optimal location can be limited by only considering a small set of clients for the MinMax objective [10] and can be limited to the vertices for the MinSum objective [5] [37]. Such nice properties for the candidate points do not apply to our problem, it is challenging to find the optimal location efficiently for KOLQ.

III. Problem definition

Given a road network $G = (V, E)$, V is a set of vertices and E is a set of edges. For each edge $e = \langle v_l, v_r \rangle$ of G, v_l is the left vertex of e and v_r is the right vertex of e. We define the distance between two locations on the road network as the network distance metric, which is represented by $d(\cdot, \cdot)$. We



(a) $G(V, E)$



(b) $KNLC$

Fig. 1: A running example

use a positive weight $w(c)$ to indicate the importance of the client c, (e.g. $w(c)$ is the population while c is a residential estate). For simplicity, we assume that each client weight is equal to 1 and the probability for visiting the i-th nearest server of each client is the same in all the examples.

Example. As shown in Figure 1(a), there is an example of a road network G with 5 vertices, namely $v_1, v_2, ..., v_5$, 6 servers, namely $s_1, s_2, ..., s_6$, and 3 clients, namely c_1, c_2 and c_3. The distance between the two end-points of the line segment is shown by the number nearby. v_1/c_1 shows that c_1 locates at v_1, and so do v_2/c_2 and v_3/s_3. Each client (server) is assigned to one edge in G.

Given a set L of class labels, each server is associated with a *class label* $l \in L$. For example, in Figure 1, $L = \{white, black\}$, where a white (black) server is represented by a white (black) dot. Servers with the same (different) label are called *similar* (*dissimilar* servers) servers, and have a collaborative (competitive) relationship. For example, in Figure 1, s_1 and s_3 are similar, but s_1 and s_2 are dissimilar. In particular, the *query label* is the class label of the new server to be built in *MAS*.

Given any point p on G, we denote the i-th nearest server of p in S by $NN_i(p, S)$, and the distance between p and $NN_i(p, S)$ is denoted as $p.dist_i$, i.e., $p.dist_i = d(p, NN_i(p, S))$. For example, in Figure 1(a), $NN_1(c_1, S) = s_1$ and $c_1.dist_1 = d(c_1, s_1) = 2$.

Assume that each client $c \in C$ may be attracted by its i-th ($1 \le i \le k$) nearest server with probability $\widetilde{P}_i$ ($0 \le \widetilde{P}_i \le 1$) such that $\sum_{i=1}^{k} \widetilde{P}_i = 1$.

Definition 1: Given a server set S and a label l of server, the class probability of a client c, $\overline{P}(c, S, l)$, is the total probability of c attracted by servers in S with label l.

TABLE I: Some notations used

Notation	Description
$NN_i(p, S)$	i-th nearest server in S for point p
$KNN(c)$	the set of k nearest servers of c
$\widetilde{P}_i$	the probability that a client is attracted by the i-th nearest server of the client
$f(NN_i(p, S))$	equals 1 if $NN_i(p, S)$ is a server similar to new server, 0 otherwise
$x.dist_i$	the distance between x and its i-th nearest server
$\overline{P}(c, S)$	$\sum_{i=1}^{k} \widetilde{P}_i \cdot f(NN_i(c, S))$
$\overline{P}(c, i)$	equals $\overline{P}(c, S \cup \{p\})$ if the new server at p becomes the i-th nearest server for c
$B(c, i)$	$w(c) \cdot \overline{P}(c, i)$
$B_o(c)$	$w(c) \cdot \overline{P}(c, S)$
$\mathcal{C}(e)$	set of clients whose KNLCs cover the edge e
$\mathcal{VC}(e)$	set of virtual clients whose KNLC's cover the edge e
vc_e	virtual client associated with edge e
C_e	set of clients on edge e
$B(vc_e, i)$	$\sum_{c \in C_e} B(c, i)$

Let $f(NN_i(c, S), l)$ return 1 if $NN_i(c, S)$ has label l and 0 otherwise, then $\overline{P}(c, S, l) = \sum_{i=1}^{k} \widetilde{P}_i \cdot f(NN_i(c, S), l)$. By default, l is the query label (the label of the new server), and we write $\overline{P}(c, S)$ and $f(NN_i(c, S))$ without specification for l. For example, in Figure 1, let's say the query label is white, then $f(NN_2(c_1, S)) = f(s_2) = 0$, since s_2 is black. For simplicity, we sometimes use a location p on the road network to represent the server located at p. Accordingly, if we build a new server at location p, $\overline{P}(c, S \cup \{p\})$ denotes the class probability of c after building the new server. $\overline{P}(c, S \cup \{p\})$ is simplified as $\overline{P}(c, i)$ when the new server at p was the i-th nearest server of c. We use both notations $\overline{P}(c, S \cup \{p\})$ and $\overline{P}(c, i)$ interchangeably in the following.

Problem: Given a road network $G = (V, E)$, a set C (S) of clients (servers with labels L) on G, a positive integer k, the visiting probabilities $\widetilde{P}_i$ ($1 \le i \le k$), and a query label l in L, the KOLQ problem with the KMaxSum objective function is to find an optimal location p_o for a new server with the given query label such that the objective $KMaxSum(p_o) = \sum_{c \in C} w(c) \cdot \overline{P}(c, S \cup \{p_o\})$ is maximized.

Intuitively, the purpose of the KMaxSum query is to find an optimal location such that the expected weighted sum of clients attracted by the servers (including the new server) with the query label is the greatest.

Example. Consider Figure 1. Given $k = 3$, $\widetilde{P}_1 = 0.5$, $\widetilde{P}_2 = 0.3$ and $\widetilde{P}_3 = 0.2$. Suppose that the query label is white. In this figure, c_1 is attracted by s_1 with $\widetilde{P}_1$, s_2 with $\widetilde{P}_2$, and s_3 with $\widetilde{P}_3$. Similarly, c_2 (c_3) is attracted by s_4 (s_6) with $\widetilde{P}_1$, s_5 (s_4) with $\widetilde{P}_2$, s_6 (s_3) with $\widetilde{P}_3$. Consider a new white server at a point p which is between p_3 and p_4 in Figure 1(b). Then, $KNN(c_1)$ includes s_1, the new white server at p and s_2. $\overline{P}(c_1, S \cup \{p\}) = \widetilde{P}_1 + \widetilde{P}_2 = 0.8$. Similarly, $\overline{P}(c_2, S \cup \{p\})$ and $\overline{P}(c_3, S \cup \{p\})$ are obtained. Then, $KMaxSum(p) = w(c_1) \cdot \overline{P}(c_1, S \cup \{p\}) + w(c_2) \cdot \overline{P}(c_2, S \cup \{p\}) + w(c_3) \cdot \overline{P}(c_3, S \cup$

$\{p\}) = 0.8 + 0.8 + 0.5 = 2.1$. By calculating the objective values of other locations, we can see that $KMaxSum(p)$ is the greatest. So, the point p is an optimal location for a new white server.

Consider a special case of the KMaxSum query in which there are only two class labels l_1 and l_2 and the parameters are set to be $k = 1$ and $\widetilde{P}_1 = 100\%$. There are no class labels in OLQ [5] [7] [10] [11]. But, the new server can be treated as having l_1 and all existing servers can be treated as having another label l_2. Then, the OLQ with the objective MaxSum is equivalent to the special case of the KMaxSum query. Thus, the KOLQ with the KMaxSum objective is a *generalization* of OLQ with the objective MaxSum.

IV. KMaxSum querying

For KMaxSum querying, we give some basic definitions in Section IV-A, a baseline algorithm is described in Section IV-B, and the main algorithm *MAS* is proposed in Section IV-C.

A. Basic Definitions

Firstly, we introduce a key concept called *k nearest location component* (*KNLC*).

Definition 2 (KNLC): Given any client c on road network G, the k nearest location component of c, $KNLC(c)$, is made up of k levels, where the i-th level, denoted by $KNLC(c, i)$, is the set of points on the edges in G with a distance of at most $c.dist_i$ from c. Formally, $KNLC(c, i) = \{p | d(p, c) \leq c.dist_i$ and p is a point on the edges of $G\}$.

We call $c.dist_i$ the *radius* of $KNLC(c, i)$. For example, in Figure 1(b), the radius of $KNLC(c_1, 1)$ is equal to $c_1.dist_1 = d(c_1, s_1) = 2$. $KNLC(c_1, 1)$ corresponds to the interval $[c_1, s_1]$ on the edge $\langle v_1, v_3\rangle$ and the interval $[c_1, p_2]$ on the edge $\langle v_1, v_2\rangle$.

We say that $KNLC(c, i_1)$ is *lower* than $KNLC(c, i_2)$ if $i_1 < i_2$; if $i_1 > i_2$, $KNLC(c, i_1)$ is *higher* than $KNLC(c, i_2)$. From the definition of *KNLC*, we can see that $KNLC(c, i_1) \subseteq KNLC(c, i_2)$, if $i_1 < i_2$. Also, $KNLC(c, k)$ includes each level of $KNLC(c)$.

[10] [11] introduce the concept of nearest location components (*NLC*) to enhance the efficiency of OLQ. For each client c, $NLC(c)$ is a set of points in G with a distance to c no more than $c.dist_1$. Intuitively, $NLC(c)$ just corresponds to the first level of $KNLC(c)$, i.e., $KNLC(c, 1)$. Thus, the concept of *KNLC* is a *generalization* of the concept of *NLC*.

Next, we introduce the concepts of boundary point and slot, which are used in our algorithms.

Definition 3 (boundary points and slots): For each client $c \in C$, we say that a point p on G is a **boundary point** of $KNLC(c)$ if $d(p, c) = c.dist_i$ $(1 \leq i \leq k)$. Given boundary points p_1, p_2, ..., p_i on an edge $e = \langle v_l, v_r\rangle$ with increasing distances from v_l. Then, we define each of the intervals (v_l, p_1), (p_1, p_2), ..., (p_{i-1}, p_i), (p_i, v_r) as a **slot**. If there is no boundary point on e, then the edge of e (i.e., the interval (v_l, v_r)) itself is a slot.

Note that a slot (excluding the end-points of the slot) contains no boundary point.

For example, in Figure 1(b), p_2, p_4, s_1 and s_2 are the boundary points of $KNLC(c_1)$. If $k = 3$, then the edge $\langle v_1, v_3\rangle$ can be divided into three slots by the boundary points s_1, s_2 and s_3, namely (v_1, s_1), (s_1, s_2) and (s_2, s_3). Here s_3 coincides with v_3. Note that the interval is not a slot because it contains a boundary point p_2.

B. The Baseline Algorithm

The first algorithm we introduce is a baseline algorithm which is based on some properties that are described by two lemmas in the following.

Lemma 1: For any two points p_1 and p_2 in a slot, $KMax-Sum(p_1) = KMaxSum(p_2)$.

Due to the space constraint, all proofs of Lemmas are in the appendix. Lemma 1 shows that any two points in a slot have the same objective value. We define the *objective value* of a slot to be that of any point in this slot.

The purpose of KMaxSum query is to find a location p on G such that $KMaxSum(p)$ is the greatest. The computation of $KMaxSum(p)$ requires the value of $\overline{P}(c, S \cup \{p\})$ for each client c, which can be computed based on Lemma 2.

Lemma 2: If p is not inside $KNLC(c)$ then $\overline{P}(c, S \cup \{p\}) = \overline{P}(c, S)$. Otherwise, if p is inside $KNLC(c)$ and the new server at p is the i-th nearest server of c, then

$$\overline{P}(c, S \cup \{p\}) = \sum_{1 \leq j < i} \widetilde{P}_j \times f(NN_j(c, S)) + \widetilde{P}_i + \sum_{i < j \leq k} \widetilde{P}_j \times f(NN_{j-1}(c, S))$$

With Lemma 1 and Lemma 2, we derive a baseline algorithm which includes three key steps. (1) The first step is to find all slots on edges. (2) The second step is to compute the objective value for each slot. We need to pick a location p inside the slot and calculate $kMaxSum(p)$, by running a naive Dijkstra's algorithm, we may get the clients attracted by the new server locates at p. We can get $kMaxSum(p)$ in $O(|V|log|V|+k|C|)$ time. (3) The third step is to select the slot or the vertex (not boundary point) with the greatest objective value and return it as the optimal location. The time complexity of these three key steps is $O(k|C||E|(|V|log|V| + k|C|))$.

For any point p on G, p is either in a slot or not. If p is not in any slot, p is a boundary point or a vertex. Since a boundary point is not considered for the new server, the baseline algorithm has computed the objective values for each possible point for the new server. Thus, the correctness of this algorithm is easy to verify.

C. The MAS Algorithm

The baseline algorithm needs to compute the objective values for each possible point. This is costly. In this section, we introduce an improved algorithm called *MAS* for the KMaxSum query. Let X_o denote the possible weighted sum of the attracted clients before the new server is built. Then, $KMaxSum(p) - X_o$ is the incremental weighted sum after the

new server is built at p. If p is outside $KNLC(c)$, then there is no increase from the client c. This means if we compute $KMaxSum(p) - X_o$ instead of $KMaxSum(p)$, then we need not consider any client c whose KNLC does not contain the location p, that is p is outside $KNLC(c)$. Moreover, we use the upper bounds for each edge of road network to reduce the number of edges to be further scanned. Then, the method of edge scanning is used to find the optimal location from the remaining edges. As shown in the experiments, the *MAS* algorithm can enhance dramatically the KMaxSum query performance.

1) Upper Bound and Fine-grained Pruning:

Definition 4: The benefit of client c with the new server, which becomes the i-th nearest server of c, is defined to be $B(c,i) = w(c) \cdot \overline{P}(c,i)$ $(1 \le i \le k)$.

Definition 5: The benefit of c before building the new server is defined to be $B_o(c) = w(c) \cdot \overline{P}(c,S)$.

Let $\mathcal{C}(e)$ denote the set of clients whose KNLCs overlap with the edge e. For each client $c \in \mathcal{C}(e)$, $t(c)$ is one level of $KNLC(c)$ that overlaps with the edge e and maximizes $B(c,t(c))$, namely $t(c) = argmax_i\{B(c,i)|KNLC(c,i)$ overlaps with $e\}$.

Definition 6: For each edge $e \in E$, we define $Upp(e) = \sum_{c\in\mathcal{C}(e)}(B(c,t(c)) - B_o(c))$.

Lemma 3: $Upp(e)$ is an upper bound on the maximum increase for the total benefit of all clients if the new server is built on e.

The computation of $Upp(e)$ for each edge e is shown in Algorithm 1. The key idea of this algorithm is to accumulate the *contribution* of each client to the upper bound for each edge overlapping with its KNLC. The main algorithm process corresponds to Lines 3 - 22. For each client c, we use Dijkstras algorithm to traverse the vertices in G in ascending order of their distances to c. For an edge e' that overlaps with $KNLC(c)$, we need to figure out with which layers of $KNLC(c)$ it overlaps. Note that by Dijkstra's algorithm, there are at most two chances to access e'(i.e. via its two vertices). Lines 10 - 12 show the first visit to e'. We know the lowest layer of $KNLC(c)$ that overlaps with e' is the i^{th} layer, so we update $Upp(e')$ in Line 11 where t is the layer higher than i and has the highest benefit. Lines 14 - 16 show the second visit to e'. We know that s is the lowest layer of $KNLC(c)$ that overlaps with e' in Line 9, and i is the highest layer, then we get the layer t' with the highest benefit between the s^{th} layer and the i^{th} layer in Line 15, and update $Upp(e')$ in Line 16. Dijkstra's algorithm in Line 5 is the main cost for Algorithm 1. Since Dijkstra's algorithm takes $O(|V|\log|V|)$ time [38], we suppose $E = O(V)$, so Algorithm 1 needs $O(|C||V|\log|V|)$ time in the worst case of traversing the whole road network.

Based on the upper bound for each edge, we propose a *fine-grained pruning* strategy as follows. Edges are examined in non-ascending order of their upper bounds. Then, the method of edge scanning, which will be introduced later, is used to find the optimal objective value for each examined edge. After that, we can prune the unexamined edges whose upper bounds are less than the largest increase of benefits for the clients found so far.

Algorithm 1: Computing the upper bound for each edge

```
   Input: G, C, k, P̃_i(1 ≤ i ≤ k) and c.dist_t(1 ≤ t ≤ k)
   Output: Upp(e) for each edge e ∈ E
 1 initialize Upp(e) ← 0 for each edge e ;
 2 for each client c ∈ C do
 3     let e be the edge that contains client c, update Upp(e) properly ;
 4     i ← 1, d ← c.dist_i, t ← argmax_{1≤j≤k} B(c, j) ;
 5     apply Dijkstra's algorithm to traverse V in ascending order of their
       distances to c ;
 6     for each vertex v in ascending order of its distance to c do
 7         if d(v, c) < d then
 8             for each edge e' adjacent to v in the form of ⟨v, v'⟩ do
 9                 Let the last updated record of e' be (c', s) ;
10                 if c' ≠ c then
11                     Upp(e') ← Upp(e') + B(c, t) − B_o(c) ;
12                     update the last update record of e' to be (c, i) ;
13                 end
14                 else
15                     t' ← argmax_{s≤j≤i} B(c, j) ;
16                     Upp(e') ← Upp(e') + B(c, t') − B(c, t) ;
17                 end
18             end
19         else
20             while i ≤ k and c.dist_i < d(v, c) do  i ← i + 1 ;
21             if i > k then break ;
22             d ← c.dist_i ; t ← argmax_{i≤j≤k} B(c, j) if t > i;
23         end
24     end
25 end
26 return Upp(e) for each edge e ;
```

2) Virtual Client and Coarse-grained Pruning: Next we introduce a *coarse-grained pruning* strategy which is based on the concept of *virtual clients*. With this strategy, we obtain a *relaxed* new upper bound $NewUpp(e)$ by less computation.

Consider the edge $e = \langle v_l, v_r\rangle$ containing some clients. The *virtual client* associated with the edge e, denoted by vc_e, is defined as follows. (1) $vc_e.dist_i = \max\{\max_{c\in C_e}\{c.dist_i - d(v_l,c)\}$, 0, $\max\{c.dist_i - d(v_r,c)\}\}$, where C_e is the set of clients on e. (2) if p is any point on the edge e, then $d(vc_e,p) = 0$, otherwise, $d(vc_e,p) = \min\{d(v_l,p), d(v_r,p)\}$. Intuitively, the whole of edge e is viewed as the virtual client vc_e.

The definitions related to a real client are adopted for a virtual client. By Definition 2, we define $KNLC(vc_e)$ based on the above $vc_e.dist_i$. Then, $KNLC(vc_e,i_1) \subseteq KNLC\ (vc_e,i_2)$ $(1 \le i_1 < i_2 \le k)$, $\cup_{i=1}^{k} KNLC(vc_e,i) = KNLC(vc_e,\ k)$. Note that if the k nearest servers of the client c are on e, $c.dist_i - d(v_l,c)$ and $c.dist_i - d(v_r,c)$ may be less than 0. Then we may have $vc_e.dist_i = 0$. Since $d(vc_e,p) = 0$ for any point p on e, in the case that $vc_e.dist_i \le 0$, the edge e is defined to be included in $KNLC(vc_e,1)$. In particular, if there is only one client c on e, $KNLC(vc_e)$ is regarded the same as $KNLC(c)$, i.e., $KNLC(vc_e,i) = KNLC(c,i)$.

Lemma 4: For any client c on the edge e, $KNLC(c,i) \subseteq KNLC(vc_e,i)$ for $1 \le i \le k$.

Lemma 4 tells us that the i-th level of $KNLC(vc_e)$ includes the i-th level of $KNLC(c)$ for each client c on the edge e. Thus, $KNLC(c) \subseteq KNLC(vc_e)$.

Different from Definition 4, the benefit of virtual client vc_e with the new server is defined as $B(vc_e,i) = \sum_{c\in C_e} \max_{i\le j\le k}\ B(c,j)$, where C_e is the set of clients on

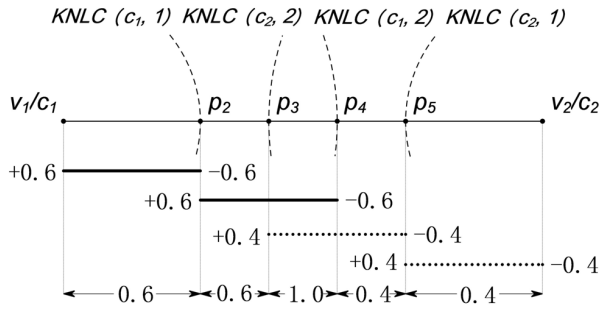


Fig. 2: An example of edge scanning

the edge e, and we can see the monotonicity of $B(vc_e, i)$(i.e. $B(vc_e, i_1) \geq B(vc_e, i_2)$ if $i_1 < i_2$). Similar to Definition 5, the benefit of virtual client vc_e without the new server is defined to be $B_o(vc_e) = \sum_{c \in C_e} B_o(c)$. Besides, we denote by $\mathcal{VC}(e)$ the set of virtual clients whose KNLCs overlap with the edge e. Similar to $t(c)$, for any $vc \in \mathcal{VC}(e)$, $t(vc) = argmax_i\{B(vc, i)|\ KNLC(vc, i)$ overlaps with $e\}$. Due to the monotonicity of $B(vc, i)$, we can see $t(vc) = \min\{i|KNLC(vc, i)$ overlaps with $e\}$.

Definition 7: For each edge $e \in E$, we define $NewUpp(e) = \sum_{vc \in \mathcal{VC}(e)} (B(vc, t(vc)) - B_o(vc))$.

The computation for $NewUpp(e)$ for each edge e is similar to the computation for $Upp(e)$ (i.e., Algorithm 1). In general, the computation of $NewUpp(e)$ for each edge e takes $O(\epsilon|V| \log |V|)$ time, where ϵ is the number of edges containing at least one client. Notice that ϵ is typically smaller than $|E|$ or $|C|$.

Lemma 5: $Upp(e) \leq NewUpp(e)$.

Lemma 5 says that $NewUpp(e)$ is a *relaxed* upper bound compared with $Upp(e)$. Similar to the fine-grained pruning discussed before, we can construct the *coarse-grained pruning* based on the new upper bound for each edge.

3) Edge Scanning: Suppose that the new server is on an edge e of G. After computing all boundary points and point intervals on e, we obtain the benefits associated with each point interval. Based on Lemma 1, by scanning each interval on e, we find the location with the optimal objective value.

Let us illustrate with an example. Consider the edge $e = \langle v_1, v_2 \rangle$ in Figure 1(b). Suppose that $k = 2$, $w(c_1) = 0.6$ and $w(c_2) = 0.4$. As shown in Figure 2, there are four boundary points p_2, p_4, p_5 and p_3 generated by $KNLC(c_1)$ and $KNLC(c_2)$ on the edge e. Then, there are four slots $[v_1, p_2]$, $(p_2, p_4]$, $[p_5, v_2]$ and $[p_3, p_5)$ among which the former two are *associated with* $B(c_1, 1) = 0.6$ and $B(c_1, 2) = 0.6$, respectively, and the latter two are associated with $B(c_2, 1) = 0.4$ and $B(c_2, 2) = 0.4$, respectively.

Let X be used to compute the optimal objective value and initialized to zero. In Figure 2, the start (end) point of each slot is marked with the symbol of "+" ("−"). The number next to each symbol is the benefit associated with the corresponding slot. When we move from v_1 to v_2, if we hit the start point of an interval, this means that we will enter the range of this interval, thus we increase X with the benefit of the interval. Otherwise, if we hit the end point of an interval, that means we will leave this interval and so we decrease X with the benefit of the interval. Firstly, we hit v_1 and enter $KNLC(c_1, 1)$, so $X = 0.6$. Next, we hit p_2, where we leave $KNLC(c_1, 1)$ and enter $KNLC(c_1, 2)$. So, $X = 0.6$. Next, we hit p_3 and enter $KNLC(c_2, 2)$ before leaving $KNLC(c_1, 2)$, which means that $X = 0.6 + 0.4 = 1$. Next, we hit p_4 and leave $KNLC(c_1, 2)$, so $X = 1 - 0.6 = 0.4$. Next, we hit p_5. Similarly, we leave $KNLC(c_2, 2)$ and enter $KNLC(c_2, 1)$, so $X = 0.4$. Finally, we reach v_2 and this scanning process is finished.

Algorithm 2: The MAS algorithm

```
Input: G, C, k, P̃_i(1 ≤ i ≤ k), c.dist_t(1 ≤ t ≤ k) and the query label
Output: The optimal location and the objective value
1  X ← 0, P ← ∅, X_o ← 0 ;
2  for each client c ∈ C do
3  |   X_o ← X_o + B_o(c) ;
4  end
5  compute the upper bound Upp(e) (NewUpp(e)) for each edge e ∈ E ;
6  sort all edges in non-ascending order of their upper bounds ;
7  ScanEdges(X, X_o, P) ;
8  return P and X ;
   procedure ScanEdges(X, X_o, P)
       for each edge e to be processed in sorted ordering do
           if Upp(e)(NewUpp(e)) < X − X_o then
               break ;
           else
               scan e to find the location with the optimal objective value X' ;
               if X' > X then
                   X ← X' ;
                   P ← {p|p is the location with X } ;
               end
           end
       end
   end procedure
```

Next, we examine the vertices of e which are not boundary points. During the edge scanning, we need to check if a vertex is included in a point interval. If so, the objective value for the vertex should be incremented with the benefit of the interval. For example, since v_1 is included in $[v_1, p_2)$, its objective value is equal to 0.6. Finally, the optimal objective value is equal to 1 and the optimal location for this example is any point in the slot (p_3, p_4).

Putting things together, Algorithm 2 is the *MAS* algorithm. Firstly, the upper bound for each edge is computed. Then, edges are sorted in non-ascending order of their upper bounds and pruned by the bounds. Finally, the optimal location and the objective value are found by scanning the remaining edges.

Complexity. In Algorithm 2, Lines 2-3 need $O(|C|)$ time and space. If $Upp(e)$ ($NewUpp(e)$) is used, Line 5 needs $O(|C||V| \log |V|)$ ($O(\epsilon|V| \log |V|)$) time. Line 5 takes $O(|V| + |E|)$ space. Line 6 needs $O(|E| \log |E|)$ time and $O(|E|)$ space. Let the number of edges that are scanned be β. Typically, $\beta \ll |E|$. In the experiments, β is at most 16 and usually smaller than 6. Then, the edge scanning procedure takes $O(\beta(|V| \log |V| + k|C| \log k|C|))$ time and $O(|V| + k|C|)$ space. Thus, the total time is $O(|C||V| \log |V| + |E| \log |E| + \beta(|V| \log |V| + k|C| \log k|C|))$ and the total space is $O(|V| + |E| + k|C|)$.

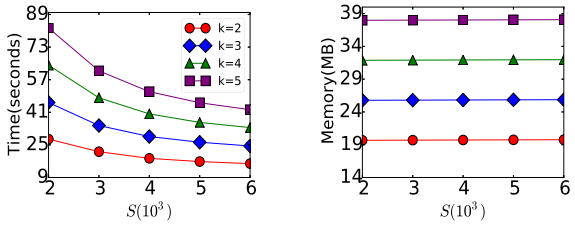
V. Empirical studies

In this section, we evaluated the performance of our proposed algorithms. The description of experiment environment and datasets are as follows.

- Hardware and platform: we run experiments on a machine with a 3.4Ghz*8 Intel Core i7-4770 CPU and 16 GB RAM, running Ubuntu 12.04 LTS Linux OS. All algorithms were implemented in C++ and compiled with GNU C++ compiler.
- Real Datasets: we use two widely used real road networks, i.e. road network *SF* (San Francisco) and *COL* (Colorado). *SF* contains 174,955 vertices and 223,000 edges. The way of generating clients and servers in SF is similar to [5] [7] [10] [11]. Specifically, we acquire a large number of real building locations in San Francisco from the *OpenStreetMap* project. The random sample sets of those real locations are used as clients and servers in *SF*. *COL* contains 435,666 vertices and 1,057,066 edges downloaded from http://www.dis.uniroma1.it/challenge9/download.shtml. As in previous works [5] and [10] on OLQ, we include synthesized clients and servers whose numbers and locations on each edge are generated randomly. The clients and the servers in road networks are stored in two separated lists.
- Settings: each client is associated with a weight which is generated randomly from a Zipf distribution with a skewness parameter $\alpha > 1$ (by default $\alpha = \infty$, which means that the weight of each client is equal to 1.), which is similar to the existing work [5]. The number $|L|$ of class labels is varied from 3 to 5 and the default value is 3. We assign randomly one class label to each server. The default value for the number $|S|$ of servers for *SF* (*COL*) is 4,000 (8,000) and the default value for the number $|C|$ of clients for *SF* (*COL*) is 400,000 (800,000). By default, $k = 3$ and $\widetilde{P}_i$ is setting inversely proportional to $c.dist_i$. For example, for a client c with $c.dist_1 = 0.2, c.dist_2 = 0.25, c.dist_3 = 1$, $\widetilde{P}_1, \widetilde{P}_2, \widetilde{P}_3$ will be $0.5, 0.4, 0.1$ respectively.

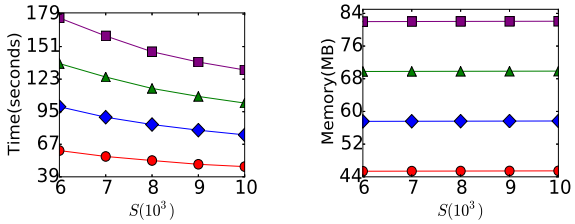
Firstly, we compare the baseline algorithm in Section IV-B with the *MAS* algorithm in Section IV-C using Upp by default. Specifically, both algorithms were executed on the same dataset of SF with 4,000 servers and 400,000 clients. The baseline algorithm takes about 15.55 hours while the *MAS* algorithm only needs about 30 seconds. The baseline algorithm is very inefficient since it computes the objective values for each possible point. In the following, we study the effects of different parameters on the *MAS* algorithm.

Effect of the number $|S|$ of servers and k: The sizes of $|S|$ and k are varied and the other parameters are set by default. The results on *SF* and *COL* are in Figure 3 and Figure 4, respectively. Note that the major time consuming parts of MAS are the KNN computation and the computation of the upper bounds for each edge. The running time of the KNN computation increases as $|S|$ increases, while the running time of the upper bound computation decreases as $|S|$ increases. However, for a large $|S|$, the upper bound computation dominates, thus, the running time decreases slowly overall, as we can see in both Figure 3(a) and Figure 4(a). Besides, when k is larger, the radius of KNLC becomes larger and thus the time of *MAS* increases. On the other hand, the memory consumption of *MAS* mainly depends on the sizes of $|V|$, $|C|$ and k. The effect of $|S|$ is small. As shown in Figure 3(b) and Figure 4(b), when $|S|$ is large, the increase of memory consumption is very small with $|S|$.



(a) time (b) storage

Fig. 3: Effect of $|S|$ and k on SF for *MAS*



(a) time (b) storage

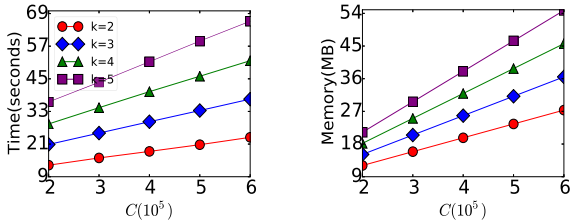
Fig. 4: Effect of $|S|$ and k on COL for *MAS*

TABLE II: Effect of k and $|L|$ on SF for *MAS*

SF	time (seconds)			memory (MB)		
	$\|L\|=3$	$\|L\|=4$	$\|L\|=5$	$\|L\|=3$	$\|L\|=4$	$\|L\|=5$
$k=2$	18.36	18.94	18.95	19.72	19.72	19.72
$k=3$	29.18	30.12	29.51	25.83	25.83	25.83
$k=4$	40.25	40.32	40.42	31.93	31.93	31.93
$k=5$	51.22	51.37	51.34	38.03	38.03	38.03

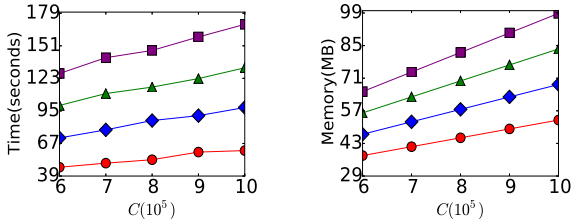
Effect of the number $|C|$ of clients and k: The results on *SF* and *COL* are in Figure 5 and Figure 6, respectively. When $|C|$ is larger, the number of KNLCs of clients becomes larger and thus the time of *MAS* increases with $|C|$. Besides, when k becomes larger, the radius of KNLCs of clients is larger and thus the time of *MAS* increases. The memory consumption of *MAS* increases with the increased sizes of $|C|$ and k.

Effect of k and the number $|L|$ of class labels: The results on *SF* and *COL* are in Table II and Table III, respectively. The time and memory consumption of *MAS* increase with the increase of k. The effect of $|L|$ is small. The memory consumption remains unchanged.



(a) time (b) storage

Fig. 5: Effect of $|C|$ and k on SF for *MAS*



(a) time (b) storage

Fig. 6: Effect of $|C|$ and k on COL for *MAS*

TABLE III: Effect of k and $|L|$ on COL for *MAS*

COL	time (seconds)			memory (MB)		
	$\|L\|=3$	$\|L\|=4$	$\|L\|=5$	$\|L\|=3$	$\|L\|=4$	$\|L\|=5$
$k=2$	52.90	54.07	54.98	45.41	45.41	45.41
$k=3$	83.71	84.80	84.97	57.62	57.62	57.62
$k=4$	115.09	116.17	115.67	69.83	69.83	69.83
$k=5$	146.59	146.59	146.74	82.03	82.03	82.03

Effect of $\widetilde{P}_i$: We use the default setting of all the parameters except for the probabilities $\widetilde{P}_i$, whose values are generated randomly. The sum of such probabilities for each client is one. We try 100 tests for the two road networks respectively. As shown in Table IV, both the time and the memory consumption of *MAS* are not sensitive to the visiting probabilities.

Effect of α: Specifically, we use the default setting of all the parameters except $w(c)$, whose values are randomly generated from a Zipf distribution with a skewness parameter α. We randomly choose the value of α in $\{2,3,4,5,6\}$, and try 100 tests for the two road networks respectively. Table V shows that the time and the memory consumption of *MAS* are not sensitive to α.

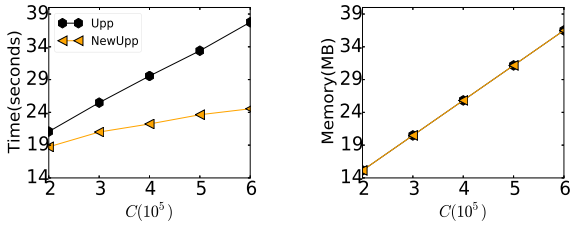
Effect of $NewUpp$: The results on the comparison of Upp and $NewUpp$ in *MAS* algorithm are in Figure 7 and Figure 8. The results show that the *MAS* algorithm with $NewUpp$ is faster. This is because the computation cost of $NewUpp$ is less than that of Upp, especially when $|C|$ is large. $NewUpp$ has comparable pruning power compared to Upp even though it is not as tight as Upp, which is another reason why we have good time performance. When $|S|$ is small, the computation cost for $KNLC$s will be larger. Since Upp is tighter than $NewUpp$, the computation cost of Upp will be larger. As

TABLE IV: Effect of $\widetilde{P}_i$

dataset	result	min	max	avg	std
SF	time	28.93	36.58	29.94	1.20
	memory	25.83	25.83	25.83	0.00
COL	time	82.53	84.67	83.09	0.30
	memory	57.62	57.62	57.62	0.00

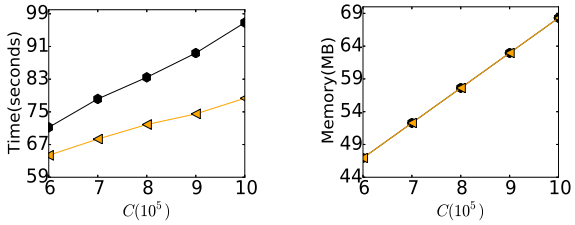
TABLE V: Effect of α

dataset	result	min	max	avg	std
SF	time	29.04	29.80	29.27	0.11
	memory	25.83	25.83	25.83	0.00
COL	time	82.55	83.94	83.17	0.22
	memory	57.62	57.62	57.62	0.00



(a) time (b) storage

Fig. 7: Effect of $|C|$ on SF for Upp and $NewUpp$



(a) time (b) storage

Fig. 8: Effect of $|C|$ on COL for Upp and $NewUpp$

shown in these figures, both Upp and $NewUpp$ have similar memory consumption. The lines for memory consumption are overlapping in these figures.

VI. Conclusion

In this paper, we study the KOLQ problem with the K-MaxSum objective function, namely the KMaxSum querying problem. We also propose algorithms for the problem. Our algorithms incorporate some new pruning techniques based on the concept of *KNLC*. We verify the performance of the algorithms on two datasets based on real world road networks of San Francisco and Colarado. Our results show that our algorithms can handle queries with reasonable time and memory. The KOLQ problem with other objective functions will be studied in our future work.

ACKNOWLEDGEMENTS

This paper is supported by the National Nature Science Foundation of China (NSFC 61572537, U1501252). Yubao Liu is the corresponding author.

REFERENCES

[1] S. Cabello, J. M. Diaz-Banez, S. Langerman, C. Seara, and I. Ventura, "Reverse facility location problems," in *CCCG*, 2005.
[2] J. Cardinal and S. Langerman, "Min-max-min geometric facility location problems," in *EWCG*, 2006.
[3] J. Krarup and P. M. Pruzan, "The simple plant location problem: Survey and synthesis," *European Journal of Operational Research*, vol. 12, no. 1, pp. 36–57, 1983.
[4] B. C. Tansel, R. L. Francis, and T. J. Lowe, "Location on networks: A survey," *Management Science*, vol. 29, no. 4, pp. 498–511, 1983.
[5] X. Xiao, B. Yao, and F. Li, "Optimal location queries in road network databases," in *ICDE*, 2011.
[6] M. de Berg, M. van Krefeld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000.
[7] B. Yao, X. Xiao, F. Li, and Y. Wu, "Dynamic monitoring of optimal locations in road network databases," *VLDB J.*, vol. 23, no. 5, pp. 697–720, 2014.
[8] J. Qi, R. Zhang, L. Kulik, D. Lin, and Y. Xue, "The min-dist location selection query," in *ICDE*, 2012.
[9] J. Qi, R. Zhang, Y. Wang, A. Y. Xue, G. Yu, and L. Kulik, "The min-dist location selection and facility replacement queries," *World Wide Web*, vol. 17, no. 6, pp. 1261–1293, 2014.
[10] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, G. Mai, and C. Long, "Efficient algorithms for optimal location queries in road networks," in *SIGMOD*, 2014.
[11] ——, "Optimal location queries in road networks," *ACM Transactions on Database Systems*, vol. 40, no. 3, p. 17, 2015.
[12] Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu, "Maxfirst for maxbrknn," in *ICDE*, 2011.
[13] J. Sankaranarayanan, H. Samet, and H. Alborzi, "Path oracles for spatial networks," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 1210–1221, 2009.
[14] H. Hu, D. L. Lee, and V. Lee, "Distance indexing on road networks," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 894–905.
[15] H. Samet, J. Sankaranarayanan, and H. Alborzi, "Scalable network distance browsing in spatial databases," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 43–54.
[16] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis, "Continuous nearest neighbor monitoring in road networks," in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 43–54.
[17] M. A. Cheema, W. Zhang, X. Lin, Y. Zhang, and X. Li, "Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 21, no. 1, pp. 69–95, 2012.
[18] D. Yan, Z. Zhao, and W. Ng, "Efficient algorithms for finding optimal meeting point on road networks," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 1–11, 2011.
[19] R. C.-W. Wong, M. T. Ozsu, A. W.-C. Fu, P. S. Yu, and L. Liu, "Efficient method for maximizing bichromatic reverse nearest neighbor," *PVLDB*, vol. 2, no. 1, pp. 1126–1137, 2009.
[20] R. C.-W. Wong, M. T. Ozsu, A. W.-C. Fu, P. S. Yu, L. Liu, and Y. Liu, "Maximizing bichromatic reverse nearest neighbor for lp-norm in two- and three-dimensional spaces," *VLDB J.*, 2011.
[21] D. Yan, R. C.-W. Wong, and W. Ng, "Efficient methods for finding influential locations with adaptive grids," in *CIKM*, 2011.
[22] Y. Liu, R. C.-W. Wong, K. Wang, Z. Li, C. Chen, and Z. Chen, "A new approach for maximizing bichromatic reverse nearest neighbor search," *Knowl. Inf. Syst.*, vol. 36, no. 1, pp. 23–58, 2013.
[23] R. Liu, A. W.-C. Fu, Z. Chen, S. Huang, and Y. Liu, "Finding multiple new optimal locations in a road network," in *SIGSPATIAL*, 2016.
[24] P. Ghaemi, K. Shahabi, J. P. Wilson, and F. B. Kashani, "A comparative study of two approaches for supporting optimal network location queries," *GeoInformatica*, vol. 18, no. 2, pp. 229–251, 2014.
[25] J. Gamper, M. H. Böhlen, and M. Innerebner, "Scalable computation of isochrones with network expiration," in *SSDBM*, 2012.
[26] Z. Xu and H.-A. Jacobsen, "Processing proximity relations in road networks," in *SIGMOD*, 2010.
[27] E. Yilmaz, S. Elbasi, and H. Ferhatosmanoglu, "Predicting optimal facility location without customer locations," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 2121–2130.
[28] Y. Du, D. Zhang, and T. Xia, "The optimal-location query," in *SSTD*, 2005.
[29] D. Zhang, Y. Du, T. Xia, and Y. Tao, "Progressive computation of the min-dist optimal-location query," in *VLDB*, 2006.
[30] M. L. Yiu, N. Mamoulis, and D. Papadias, "Aggregate nearest neighbor queries in road networks," *IEEE Trans. Knowl. Data Eng*, vol. 17, no. 6, pp. 820–833, 2005.
[31] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," *ACM Trans. Database Syst.*, vol. 30, no. 2, pp. 529–576, 2005.
[32] H. G. Elmongui, M. F. Mokbel, and W. G. Aref, "Continuous aggregate nearest neighbor queries," *GeoInformatica*, vol. 17, no. 1, pp. 63–95, 2013.
[33] F.Korn and S.Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *SIGMOD*, 2000.
[34] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Norvag, "Monochromatic and bichromatic reverse top-k queries," *IEEE Trans. Knowl. Data Eng*, vol. 23, no. 8, pp. 1215–1229, 2011.
[35] T. Bernecker, T. Emrich, H.-P. Kriegel, M. Renz, S. Zankl, and A. Zufle, "Efficient probabilistic reverse nearest neighbor query processing on uncertain data," *PVLDB*, vol. 4, no. 10, pp. 669–680, 2011.
[36] M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei, "Probabilistic reverse nearest neighbor queries on uncertain data," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 4, pp. 550–564, 2010.
[37] L. Xu, G. Mai, Z. Chen, Y. Liu, and G. Dai, "Minsum based optimal location query in road networks," in *International Conference on Database Systems for Advanced Applications*. Springer, 2017, pp. 441–457.
[38] E. W. Dijkstra, "A note on two problems in connexion with graphs," *NUMERISCHE MATHEMATIK*, vol. 1, no. 1, pp. 269–271, 1959.

APPENDIX

We introduce the proofs of lemma 1-5 as follows.

Proof of Lemma 1. Let p_1 and p_2 be in the same slot ξ. For any client c, there are three cases.

Case 1: $KNLC(c)$ cannot cover ξ. Namely, the new server established at p_1 or p_2 cannot affected the client c. Thus, $f(NN_i(c, S \cup \{p_1\})) = f(NN_i(c, S)) = f(NN_i(c, S \cup \{p_2\}))$ for each $1 \leq i \leq k$.

Case 2: $KNLC(c)$ partly covers ξ. For this case, there exists a point p in $\mathcal{S}$ which is a boundary point. This contradicts the fact ξ is a slot.

Case 3: $KNLC(c)$ entirely covers ξ. For this case, we assume that the k nearest servers of c are s_1, s_2, ..., s_k before building the new server. Assume that the new server built at the locations $p_1(p_2)$ becomes the t-th (t'-th) nearest server of client c. We assume that t is not equal to t' and $t < t'$ without loss of generality. Since the new server at p_1 is the t-nearest server of c, the original t-nearest server of c, s_t $(1 \leq t < k)$ should be the $(t+1)$-nearest server of c after the new server is built. Then we have $d(c, p_1) < d(c, s_t) \leq d(c, p_2)$. Thus, we can find a point p in the segment [p1,p2] such that $d(c, p) = d(c, s_t) = c.dist_t$. This means p is a boundary point of $KNLC(c)$, which contracts to the fact that B is a slot. Thus, we have $t = t'$. For $i < t$, we have $NN_i(c, S \cup \{p_1\}) = NN_i(c, S \cup \{p_2\}) = s_i$ and then $f(NN_i(c, S \cup \{p_1\})) = f(NN_i(c, S \cup \{p_2\}))$. For $i > t$, we have $NN_i(c, S \cup \{p_1\}) = NN_i(c, S \cup \{p_2\}) = s_{i-1}$ and then

$f(NN_i(c, S \cup \{p_1\})) = f(NN_i(c, S \cup \{p_2\}))$. For $i = t$, we have $NN_i(c, S \cup \{p_1\}) = p_1$ and $NN_i(c, S \cup \{p_2\}) = p_2$. Then $f(NN_i(c, S \cup \{p_1\})) = f(NN_i(c, S \cup \{p_2\})) = 1$. Thus, $f(NN_i(c, S \cup \{p_1\})) = f(NN_i(c, S \cup \{p_2\}))$ for each $1 \leq i \leq k$.

$KMaxSum(p_1) - KMaxSum(p_2) = \sum_{c \in C} w(c) \cdot \sum_{i=1}^{k} \widetilde{P}_i \cdot (f(NN_i(c, S \cup \{p_1\}) - f(NN_i(c, S \cup \{p_2\})) = 0$.

This lemma holds.

Proof of Lemma 2. Let $S' = S \cup \{p\}$. If p is not inside $KNLC(c)$ then the client c is not affected by the new server at p. It is easy to know $\overline{P}(c, S') = \overline{P}(c, S)$. Otherwise p is inside $KNLC(c)$ and the new server is the i-th nearest server of c ($1 \leq i \leq k$). For any $j < i$, $NN_j(c, S') = NN_j(c, S)$. For any $j > i$, $NN_j(c, S') = NN_{j-1}(c, S)$. $\overline{P}(c, S') = \sum_{1 \leq j < i} \widetilde{P}_j \times f(NN_j(c, S')) + \widetilde{P}_i \times f(NN_i(c, S')) + \sum_{i < j \leq k} \widetilde{P}_j \times f(NN_j(c, S')) = \sum_{1 \leq j < i} \widetilde{P}_j \times f(NN_j(c, S)) + \widetilde{P}_i + \sum_{i < j \leq k} \widetilde{P}_j \times f(NN_{j-1}(c, S))$.

The lemma holds.

Proof of Lemma 3. Given any client $c \in \mathcal{C}(e)$, we assume that there are j levels of $KNLC(c)$ covering the edge e, namely $KNLC\ (c, i_1)$, $KNLC(c, i_2)$, ... , $KNLC(c, i_j)$, where $\overline{P}(c, i_1) \geq \overline{P}(c, i_2) \cdots \geq \overline{P}(c, i_j)$. Then $t(c) = i_1$. By Definition 4 , $B(c, i_1) \geq B(c, i_2)$ ... $\geq B(c, i_j)$. If the new server is on e, $B(c, t(c))$ is the greatest benefit for the client c with the new server. Since $B_o(c)$ is the benefit for c without the new server, the largest increase of the benefit for c is at most equal to $B(c, t(c)) - B_o(c)$.

Thus, the lemma holds.

Proof of Lemma 4. For any point $p \in KNLC(c, i)$, there are two cases. Case 1: p is on the edge e. Case 2: p is not on the edge e. For Case 1, since p is on e, and e is included in $KNLC(vc_e, 1)$. Thus, p is in $KNLC(vc_e, 1)$. This lemma holds. For Case 2, since p is not on the edge $e = \langle v_l, v_r \rangle$, $d(c, p) = d(v_l, c) + d(v_l, p)$ or $d(c, p) = d(v_r, c) + d(v_r, p)$. W.l.o.g., suppose that $d(c, p) = d(v_l, c) + d(v_l, p)$. Since $d(c, p) \leq c.dist_i$, $d(v_l, c) + d(v_l, p) \leq c.dist_i$, $d(v_l, p) \leq c.dist_i - d(v_l, c) \leq vc_e.dist_i$. Then, $d(vc_e, p) = \min\{d(v_l, p), d(v_r, p)\} \leq vc_e.dist_i$.

Thus, $p \in KNLC\ (vc_e, i)$. This lemma holds.

Proof of Lemma 5. Let $\mathcal{C}(e, e')$ denote the set of clients on the edge e' whose KNLCs overlap with the edge e. Let $C_{e'}$ denote the set of clients on the edge e'. Let E' denote the set of edge e' where $\mathcal{C}(e, e') \neq \emptyset$.

(1) For any client $c \in \mathcal{C}(e, e')$, c is on e' and $KNLC(c)$ overlaps with the edge e. Since c is on e', $KNLC(c) \subseteq KNLC(vc_{e'})$ by Lemma 4. Then, $KNLC(vc_{e'})$ overlaps with the edge e and $vc_{e'} \in \mathcal{VC}(e)$. Then, for any edge $e' \in E'$, $vc_{e'} \in \mathcal{VC}(e)$.

(2)It is easy to see that for any $1 \leq i \leq k$, $B(c, i) - B_o(c) \geq 0$.

(3) For any client $c \in C_{e'}$, $KNLC(c, t(c))$ overlaps with e. Since $KNLC(c, t(c)) \subset KNLC(vc_{e'}, t(c))$, $KNLC(vc_{e'}, t(c))$ overlaps with e, thus, $t(vc_{e'}) \leq t(c)$. Then, $t(vc_{e'}) \leq min\{t(c) |\ c \in C_{e'}\}$.

$$
\begin{aligned}
Upp(e) &= \sum_{c \in \mathcal{C}(e)} B(c, t(c)) - B_o(c) \\
&= \sum_{e' \in E'} \sum_{c \in \mathcal{C}(e, e')} B(c, t(c)) - B_o(c) \\
&\leq \sum_{vc_{e'} \in \mathcal{VC}(e)} \sum_{c \in \mathcal{C}(e, e')} B(c, t(c)) - B_o(c) \\
&\leq \sum_{vc_{e'} \in \mathcal{VC}(e)} \sum_{c \in C_{e'}} B(c, t(c)) - B_o(c) \\
&\text{(by } C(e, e') \subset C_{e'} \text{ and the above (2))} \\
&\leq \sum_{vc_{e'} \in \mathcal{VC}(e)} \sum_{c \in C_{e'}} \max_{t(c) \leq i \leq k} B(c, i) - B_o(c) \\
&\leq \sum_{vc_{e'} \in \mathcal{VC}(e)} \sum_{c \in C_{e'}} \max_{t(vc_{e'}) \leq i \leq k} B(c, i) - B_o(c) \\
&= \sum_{vc_{e'} \in \mathcal{VC}(e)} B(vc_{e'}, t(vc_{e'})) - B_o(vc_{e'}) \\
&= NewUpp(e)
\end{aligned}
$$

Thus, the lemma holds.