Front. Comput. Sci., 2021, 15(2): 152606
https://doi.org/10.1007/s11704-020-9279-6

RESEARCH ARTICLE

Optimal location query based on *k* nearest neighbours

Yubao LIU (✉)[1,2], Zitong CHEN[3], Ada Wai-Chee FU[3], Raymond Chi-Wing WONG[4], Genan DAI[1]

1 School of Data and Computer Science, Sun Yat-Sen University, Guangzhou 510006, China
2 Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou 510006, China
3 Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong 999077, China
4 Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong 999077, China

Abstract Optimal location query in road networks is a basic operation in the location intelligence applications. Given a set of clients and servers on a road network, the purpose of optimal location query is to obtain a location for a new server, so that a certain objective function calculated based on the locations of clients and servers is optimal. Existing works assume 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 relax these assumptions and consider the *k* nearest neighbours (KNN) of clients. We introduce the problem of KNN-based optimal location query (KOLQ) which considers the *k* nearest servers of clients and labeled servers. We also introduce a variant problem called relocation KOLQ (RKOLQ) which aims at relocating an existing server to an optimal location. Two main analysis algorithms are proposed for these problems. Extensive experiments on the real road networks illustrate the efficiency of our proposed solutions.

Keywords optimal location query, *k* nearest neighbours, road network

1 Introduction

Location intelligence is the process of deriving meaningful insight from geospatial data relationships to solve a particular problem (please see Wikipedia). By Google Map and other online information sources, nearly everyone is familiar with a multitude of consumer uses for location data, such as, getting directions and finding restaurants, hotels, and gas stations in a particular geographic area. Moreover, location data is already and will continue to be a growing component of all business data. Location intelligence is becoming a core part of most enterprise business' strategies. At present, lots of location intelligence tools have been developed. A fast response is expected in an interactive setting of these tools. At the same time, many mobile devices with limited memory are installed with various mobile applications for location intelligence.

Location analysis also called optimal location query, is a basic operation in the location intelligence applications such as location planning, location-based service, and profile-based marketing [1–13]. In this paper, we consider location analysis in a road network that is an important setting for location intelligence applications.

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. Recently, researchers are paying attention to this problem because of its broad applications. However, there are some limitations on existing work.

For example, consider a manager named Tom who runs the McDonald's in Guangzhou city. He plans to build a new branch of McDonald's and wants to maximize the benefit of all branches of McDonald's (which includes the new branch). The benefit is measured by the expected number of attracted clients. Note that there are other fast food brands such as KFC which have a competitive relationship with McDonald's. The existing work cannot be used to find such an optimal location for the new branch of McDonald's. This is because *class labels* such as the fast food brands KFC and McDonald's are not given to the servers. In addition to the fast food chains, there are lots of goods which are associated with different brands, such as Huawei and Xiaomi mobile phones. The goods with the same brand may have a collaborative relationship, but goods with different brands have a competitive relationship. On the other hand, as shown in [14], the fast food consumption pattern shows that clients care about time and also taste. Thus, clients may choose the fast food servers that are not too far. Clearly, the *k* nearest neighbours (KNN) of clients can provide better opportunity to choose. But, this case cannot be supported by the existing work. This is because clients are assumed to visit the servers nearest to them.

Motivated by the above observations, in this work, we allow

Received July 30, 2019; accepted November 31, 2019

E-mail: liuyubao@mail.sysu.edu.cn

* The initial version of this work has been included in the proceedings of MDM 2019 as a regular research paper [15]

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 ith 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.

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 query (KOLQ) in road networks with the KMaxSum objective is called *KMaxSum querying*. An application scenario for KOLQ problem is shown in example application 1 as follows.

Example application 1 Let C and S denote a set of residential estates and a set of branches of fast food service restaurant chains, respectively. There are different brands of fast food chains (e.g., KFC and McDonald's) and each chain consists of a lot of service branches. Assume that customers in C would visit any of their k nearest branches of restaurants in S, with a certain probability for each of such branches. A KMaxSum querying may be used to return an optimal location for a new branch of a certain fast food chain, such that this fast food chain (including the new branch) can attract the greatest number of customers.

In addition, we investigate a variant of KOLQ problem, which is useful in reality as shown in example application 2 below.

Example application 2 Consider the example application 1 again. Due to *limited* resources, a fast food chain may want to keep the same number of service branches, but move an existing service branch to a new location in order to maximize the *KMaxSum* objective (i.e., the expected number of customers attracted by the branches of the restaurant).

We refer to such a relocation problem as relocation KNN-based optimal location query (RKOLQ) and call the corresponding querying *RKMaxSum* querying. Note that the solution involves choosing both the new location and the existing server to be moved.

To the best of our knowledge, this is the first study considering both server labels and KNN servers in road networks. Since the existing solutions are not applicable to our problem, we need to design new algorithms for this problem. The contributions of this paper can be summarized as follows.

- We propose the problem of KOLQ with the KMaxSum objective, which is more natural compared with the existing OLQ. 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*).
- We propose the relocation problem of RKOLQ. An algorithm called *RMAS* is introduced for Relocation KMaxSum querying, which utilizes pruning techniques for reducing the servers that are examined.
- 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.

Compared with the initial version [15], the Relocation KMaxSum query and its related experiments are the new contribution. The difficulty for RKOLQ is that no heuristic strategy can be applied to decide which existing server should be relocated. So we have to enumerate the server to be relocated. The contribution is that instead of calling the *MAS* algorithm multiple times, we save some intermediate results to reduce lots of duplicated computation.

The rest of this paper is organized as follows. Section 2 reviews the related work and point out the difficulties of adopting existing OLQ methods to KOLQ. Section 3 gives the problem definition. Section 4 and Section 5 introduce our proposed query algorithms. Section 6 reports on the empirical study and Section 7 concludes the paper.

2 Related work

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

Various topics on road network are studied in recent years. Some works focus on the efficiency of the shortest paths calculation, [16] 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. [17] 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 [18–20]. Among all kinds of querying, the optimal meeting point (OMP) query [21], is also a location determination problem, but with target different 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. J. Qi et al. [22] study the Min-dist location problem and Huang et al. [23] study the top-k most influential location selection query , they both find an optimal facility from a candidate set, and propose branch and bound algorithms making use of geometric properties. However, we do not have a candidate set to limit the search space, and their techniques on Euclidean space are not suitable on road network, neither the techniques in [24] aiming at metric spaces instead of Euclidean spaces.

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 [25] and some extensions of this algorithm were studied in [26]. An approximation approach was introduced for the MaxBRNN problem in [27]. An improved algorithm for MaxBRNN was given in [28].

Zhou et al. [29] 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]. [30] first studied the exact solution for the OLQ problem with multiple new servers. [31] 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 [32], which is similar to the OLQ query. Xu and Jacobsen [33] studied the problem of proximity queries among sets of moving objects in road networks. The OLQ without the customer locations was studied in [34].

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 [35] was proposed to find an optimal location instead of an optimal region for the L_1-norm space. The algorithm in [36] 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 [37–39], 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 [40] and the reverse top-k problem [41] are similar to ours in some way, their algorithms cannot handle road network. The probabilistic reverse nearest neighbor queries studied in [42,43] 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, 44]. Such nice properties for the candidate points do not apply to our problem, it is challenging to find the optimal location efficiently for KOLQ.

3 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 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 ith nearest server of each client is the same in all the example. Table 1 lists the notations used in this paper.

Example 1 As shown in Fig. 1(a), there is an example of a road network G with five vertices, namely $v_1, v_2, \ldots, v_5$, six servers, namely $s_1, s_2, \ldots, s_6$, and three clients, namely c_1, c_2

Table 1 Some notations used

Notation	Description
$NN_i(p, S)$	ith 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 ith 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 ith 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 ith nearest server for c
$B(c, i)$	$w(c) \cdot \overline{P}(c, i)$
$B_o(c)$	$w(c) \cdot \overline{P}(c, S)$
$C(e)$	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
$B(vc_e, i)$	$\sum_{c \in C_e} B(c, i)$

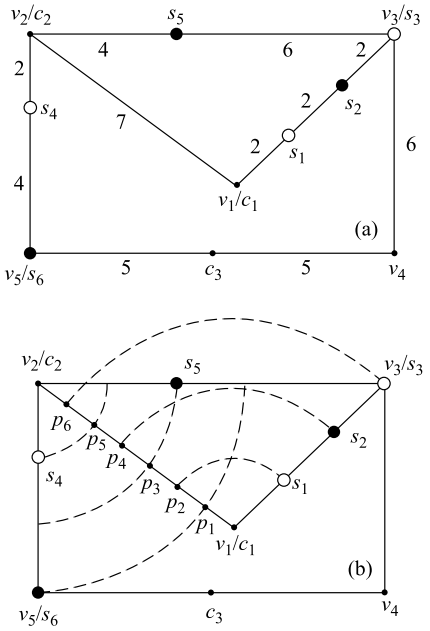

Fig. 1 A running example. (a) $G(V, E)$; (b) *KNLC*

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 Fig. 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, and have a collaborative (competitive) relationship. For example, in Fig. 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* or the existing server to be moved in *RMAS*.

Given any point p on G, we denote the ith 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 Fig. 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 ith $(1 \leqslant i \leqslant k)$ nearest server with probability $\widetilde{P}_i$ $(0 \leqslant \widetilde{P}_i \leqslant 1)$ such that $\sum_{i=1}^{k} \widetilde{P}_i = 1$. In practice, we can set the probability to be inversely proportional to the distance among the client and its ith nearest server.

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.

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 Fig. 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 ith nearest server of c. We use both notations $\overline{P}(c, S \cup \{p\})$ and $\overline{P}(c, i)$ interchangeably in the following.

Problem 1 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 $\widetilde{P}_i$ $(1 \leqslant i \leqslant k)$, 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 2 Consider Fig. 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 Fig. 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.

The relocation variant of KMaxSum query is to find an optimal location and an existing server such that if we move this server to the optimal location found, we can maximize the KMaxSum objective.

Problem 2 Given a road network $G = (V, E)$, a set C (S) of clients (servers with labels L) on G, a weight $w(c)$ for each $c \in C$, a positive integer k, the visiting probabilities $\widetilde{P}_i$ $(1 \leqslant i \leqslant k)$, a query label l in L, the variant of KOLQ with the KMaxSum objective function is to find an optimal location p_o and an existing server s_j with the given query label such that the objective $KMaxSum(s_j, p_o) = \sum_{c \in C} w(c) \cdot \overline{P}(c, S - \{s_j\} \cup \{p_o\})$ is maximized.

Example 3 Consider Fig. 1. Given $C = c_1, c_2$, $k = 2$, $\widetilde{P}_1 = 0.6$ and $\widetilde{P}_2 = 0.4$ and white query label. In this example, the existing white server s_3 should be moved since it attracts no clients. Let us move s_3 to a point p between p_3 and p_4 in Fig. 1(b). Then, $KMaxSum(s_3, p) = w(c_1) \cdot \overline{P}(c_1, S - \{s_3\} \cup \{p\}) + w(c_2) \cdot \overline{P}(c_2, S - \{s_3\} \cup \{p\}) = 1 + 1 = 2$. Obviously, $KMaxSum(s_3, p)$ is the greatest. So, moving s_3 to p is an opti-

mal solution.

4 KMaxSum querying

For KMaxSum querying, we give some basic definitions in Section 4.1, a baseline algorithm is described in Section 4.2, and the main algorithm *MAS* is proposed in Section 4.3.

4.1 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 ith 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, $KNLC(c, i) = \{p | d(p, c) \leqslant c.dist_i$ and p is a point on the edges of $G\}$.

We call $c.dist_i$ the *radius* of $KNLC(c, i)$. For example, in Fig. 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 $d(p, c) = c.dist_i$ $(1 \leqslant i \leqslant k)$. Given boundary points $p_1, p_2, \ldots, p_i$ 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), $(p_1, p_2), \ldots, (p_{i-1}, p_i), (p_i, v_r)$ 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 Fig. 1(b), p_2, p_4, s_1 and s_2 are 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.

4.2 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, $KMaxSum(p_1) = KMaxSum(p_2)$.

Proof 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 $1 \leqslant i \leqslant k$.

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 $s_1, s_2, \ldots, s_k$ before building the new server. Assume that the new server built at the locations $p_1(p_2)$ becomes the tth (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 $(1 \leqslant t < k)$ should be the $(t + 1)$-nearest server of c after the new server is built. Then we have $d(c, p_1) < d(c, s_t) \leqslant d(c, p_2)$. 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 $1 \leqslant i \leqslant k$. $KMaxSum(p_1) - KMaxSum(p_2) = \sum_{c \in C} w(c) \cdot \sum_{i=1}^{k} \widetilde{P}_i \cdot (f(NN_i(c, S \cup \{p_1\}) - f(NN_i(c, S \cup \{p_2\})) = 0$.

This lemma holds. □

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 ith nearest server of c, then

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

Proof 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 ith nearest server of c $(1 \leqslant i \leqslant k)$. For any $j < i$, $NN_j(c, S') = NN_j(c, S)$. For any $j > i$, $NN_j(c, S') = NN_{j-1}(c, S)$. $\overline{P}(c, S') = \sum_{1 \leqslant j < i} \widetilde{P}_j \times f(NN_j(c, S')) + \widetilde{P}_i \times f(NN_i(c, S')) + \sum_{i < j \leqslant k} \widetilde{P}_j \times f(NN_j(c, S')) = \sum_{1 \leqslant j < i} \widetilde{P}_j \times f(NN_j(c, S)) + \widetilde{P}_i + \sum_{i < j \leqslant k} \widetilde{P}_j \times f(NN_{j-1}(c, S))$. The lemma holds. □

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.

4.3 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.

4.3.1 Upper bound and fine-grained pruning

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

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 $C(e)$ denote the set of clients whose KNLCs overlap with the edge e. For each client $c \in C(e)$, $t(c)$ 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 $Upp(e) = \sum_{c \in C(e)}(B(c, t(c)) - B_o(c))$.

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.*

Proof Given any client $c \in C(e)$, we assume that there are j levels of $KNLC(c)$ covering the edge e, namely $KNLC(c, i_1)$, $KNLC(c, i_2), \ldots, KNLC(c, i_j)$, where $\overline{P}(c, i_1) \geqslant \overline{P}(c, i_2) \cdots \geqslant \overline{P}(c, i_j)$. Then $t(c) = i_1$. By Definition 4, $B(c, i_1) \geqslant B(c, i_2) \cdots \geqslant B(c, i_j)$. 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. □

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 3–18. For each client c, we use Dijkstra's 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 9–11 show the first visit to e'. We know the lowest layer of $KNLC(c)$ that overlaps with e' is the ith layer, so we update $Upp(e')$ in Line 10 where t is the layer higher than i and has the highest benefit. Lines 12–14 show the second visit to e'. We know that s is the lowest layer of $KNLC(c)$ that overlaps with e' in Line 8, and i is the highest layer, then we get the layer t' with the highest benefit between the sth layer and the ith layer in Line 13, and update $Upp(e')$ in Line 14. Dijkstra's algorithm is the main cost for Algorithm 1. Since Dijkstra's algorithm takes $O(|V| \log |V|)$ time [45], 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, P̄_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, dist ← c.dist_i, t ← argmax_{1⩽j⩽k} B(c, j);
5      for each vertex v in ascending order of its distance to c do
6          if d(v, c) < dist then
7              for each edge e' adjacent to v in the form of ⟨v, v'⟩ do
8                  Let the last updated record of e' be (c', s);
9                  if c' ≠ c then
10                     Upp(e') ← Upp(e') + B(c, t) − B_o(c);
11                     update the last update record of e' to be (c, i);
12                 else
13                     t' ← argmax_{s⩽j⩽i} B(c, j);
14                     Upp(e') ← Upp(e') + B(c, t') − B(c, t);
15         else
16             while i ⩽ k and c.dist_i < d(v, c) do
               i ← i + 1;
17             if i > k then break;
18             dist ← c.dist_i; t ← argmax_{i⩽j⩽k} B(c, j);
19 return Upp(e) for each edge e;
```

4.3.2 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, $KNLC(vc_e, i_1) \subseteq KNLC\ (vc_e, i_2)$ $(1 \leqslant i_1 < i_2 \leqslant k)$, $\cup_{i=1}^{k} KNLC(vc_e, i) = KNLC(vc_e, k)$. 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 $vc_e.dist_i \leqslant 0$, 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 $1 \leqslant i \leqslant k$.

Proof 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 $d(c, p) \leqslant c.dist_i$, $d(v_l, c) + d(v_l, p) \leqslant c.dist_i$, $d(v_l, p) \leqslant c.dist_i - d(v_l, c) \leqslant vc_e.dist_i$. Then, $d(vc_e, p) = \min\{d(v_l, p), d(v_r, p)\} \leqslant vc_e.dist_i$. Thus, $p \in KNLC(vc_e, i)$. This lemma holds. □

Lemma 4 tells us that the ith level of $KNLC(vc_e)$ includes the ith 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 $B(vc_e, i) = \sum_{c \in C_e} \max_{i \leqslant j \leqslant k} B(c, j)$, where C_e is the set of clients on the edge e, and we can see the monotonicity of $B(vc_e, i)$(i.e., $B(vc_e, i_1) \geqslant B(vc_e, i_2)$ 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 $Upp(e) \leqslant NewUpp(e)$.

Proof Let $C(e, e')$ 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 $C(e, e') \neq \emptyset$.

(1) For any client $c \in C(e, e')$, 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 $1 \leqslant i \leqslant k$, $B(c, i) - B_o(c) \geqslant 0$.

(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, $t(vc_{e'}) \leqslant t(c)$. Then, $t(vc_{e'}) \leqslant min\{t(c) | c \in C_{e'}\}$.

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

□

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.

4.3.3 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 Fig. 1(b). Suppose that $k = 2$, $w(c_1) = 0.6$ and $w(c_2) = 0.4$. As shown in Fig. 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 Fig. 2, the start (end) point of each slot is marked with the symbol of "+" ("−"). The number next to

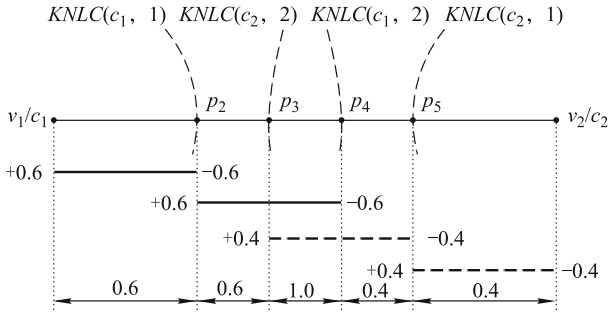

Fig. 2 An example of edge scanning

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.

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 4 needs $O(|C||V|\log|V|)$ ($O(\epsilon|V|\log|V|)$) time. Line 4 takes $O(|V| + |E|)$ space. Line 5 needs $O(|E|\log|E|)$ time and $O(|E|)$ space. Let the number of edges that are scanned be β. Typically, $\beta \ll |E|$. Lines 6–13 are the edge scanning procedure. 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 $O(|V| + |E| + k|C|)$.

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  compute the upper bound Upp(e) (NewUpp(e)) for each edge e ∈ E;
5  sort all edges in non-ascending order of their upper bounds;
6  for each edge e to be processed in sorted ordering do
7  |  if Upp(e)(NewUpp(e)) < X − X_o then
8  |  |  break;
9  |  else
10 |  |  scan e to find the location with the optimal objective value X′;
11 |  |  if X′ > X then
12 |  |  |  X ← X′;
13 |  |  |  P ← {p|p is the location with X };
14 return P and X;
```

5 The relocation KMaxSum query

In this section, we study the problem of Relocation KMaxSum querying, a variant of KMaxSum. We present a baseline algorithm and then the further enhancements.

The purpose of the query variant is to move an existing server to an optimal location such that the KMaxSum objective can be achieved. Two key issues need to be addressed. One issue is which server is selected from the existing servers and the other is where the selected server is moved to from its original location.

A simple baseline algorithm can be described as follows:

- For each server $s \in S$ with the query label, we attempt to delete s from S and obtain a remaining set of servers S_r.
- Then, the *MAS* algorithm is executed on S_r to attain an optimal location p_s and the corresponding objective value o_s, which are associated with the deletion of server s.
- After the deletion of each server in S is attempted, we select the greatest objective value and the corresponding optimal location.

Intuitively, a server which attracts a few clients may be the server to be moved, while a server which attracts many clients may be pruned. However, we observe that a server which attracts the most number of clients can also be the server to be moved. For example, consider a simple road network with only one edge, from left to right are $s_2, c_2, s_3, c_3, c_1, s_1, c_4$, and the distances between two neighbors are 1. s_1, s_2 are KFC and s_3 is McDonald's. Consider $k = 1$ and the query label is KFC, the best solution is moving s_1 to the locations between s_3 and c_1, since by the relocation, all clients will be attracted by KFC. However, s_1 is the server which attracts the most clients at the very beginning.

Since it is difficult to prune some servers, we try to improve the baseline algorithm by focusing on how to improve the processing for each server. Notice that $c.dist_i$ $(1 \leqslant i \leqslant k)$ for each client c and $Upp(e)$ for each edge e are computed from scratch for each S_r. This may not be efficient. In the following, we propose two enhancement strategies: incremental update for both $c.dist_i$ and $Upp(e)$.

Firstly, we introduce the incremental update for $c.dist_i$. We pre-compute $KNN_+(c)$, which is the set of $k+1$ nearest servers of c in S. Then for a client c, once the deleted server is in $KNN(c)$, we can easily get its new $KNN(c)$.

Secondly, we introduce the incremental update of $Upp(e)$. When s is deleted, consider two cases. Case 1: s is not in $KNN(c)$. Case 2: s is in $KNN(c)$. For Case 1, the contribution of c remains unchanged. For Case 2, after s is deleted, if

$s = NN_i(c, S)$, then for $i \leqslant j \leqslant k$, the original $(j+1)$th nearest server of c becomes the jth nearest server of c, and it should then be associated with the visiting probability $\widetilde{P}_j$ instead of the original $\widetilde{P}_{j+1}$. Thus, the contribution of c may be changed. Therefore, the contribution of c only needs to be incrementally updated for Case 2. For example, consider the examined server s_1 in Fig. 1. Since s_1 is in $KNN(c_1)$ but not in $KNN(c_2)$ and $KNN(c_3)$, we just need to examine c_1. Since $KNLC(c_1)$ covers the edge $e_1 = \langle v_1, v_2\rangle$ and $e_2 = \langle v_1, v_3\rangle$, c_1 contributes to both $Upp(e_1)$ and $Upp(e_2)$. Consider $Upp(e_1)$. The original contribution of c_1 is equal to $B(c_1, 1) - B_o(c_1) = 0.4$. After s_1 is deleted, the new contribution of c_1 becomes $B(c_1, 1) - B_o(c_1) = 0.2$. Then, the incremental contribution of c_1 is $\delta = -0.2$. Thus, $Upp(e_1)$ is updated to be $Upp(e_1) + \delta$.

The *RMAS* algorithm is given in Algorithm 3. $KNN_+(c)$ for each client c and $Upp(e)$ for each edge e are computed in advance in Line 2. It computes $Upp(e)$ for each edge e in Line 3. Then, each server $s_i \in S$ with the query label is iteratively examined. At each iteration, we compute X_{r_i}, the weighted sum of clients attracted by the remaining servers S_{r_i} in Lines 5–7, rollback the upper bound of all edges in Lines 8–9, and use the incremental update to get both $KNLC(c)$ and $Upp(e)$ in Lines 10–13, then sort the edges by $Upp(e)$ in non-ascending order in Line 14 and scan edges in Line 15. If the current best value X is updated in the edge scanning process, then we need to update X, the choice of server to be moved s', and record the optimal location in Line 17.

Complexity We are interested in the time complexity T' of running all incremental updates, i.e., the total running time of Lines 11–13. Let C_l be the set of clients attracted by the servers with the query label l say S_l, then for a client $c \in C_l$, it can appear in Line 11 at most k times. Thus, Line 11 will be run at most $k|C_l|(\leqslant k|C|)$ times, and $T' = O(k|C|(k + |V|log|V|))$, which is independent of the number of servers with the query label.

Algorithm 3 The RMAS algorithm

Input: $G, S, C, k, \widetilde{P}_i(1 \leqslant i \leqslant k), c.dist_t(1 \leqslant t \leqslant k)$ and the query label
Output: The optimal location and objective value and the selected server
1 $\mathcal{P} \leftarrow \emptyset, s' \leftarrow 0, X \leftarrow 0$;
2 find $KNN_+(c)$ and $c.dist_j$ $(j = 1, 2, \ldots, k+1)$ for each client $c \in C$;
3 compute $Upp(e)$ for each edge $e \in E$ by using $c.dist_t$ $(t = 1, 2, \ldots, k)$ and save it as $Upp'(e)$;
4 **for** each server $s_i \in S$ with the query label **do**
5 $S_{r_i} = S - \{s_i\}, X_{s_i} \leftarrow 0$;
6 **for** each client $c \in C$ **do**
7 $X_{s_i} \leftarrow X_{s_i} + B_o(c)$ based on S_{r_i};
8 **for** each edge $e \in E$ **do**
9 $Upp(e) \leftarrow Upp'(e)$;
10 **for** each client c attracted by s **do**
11 compute $KNN(c)$ and $c.dist_i$ $(i = 1, 2, \ldots, k)$ for S_{r_i} based on $KNN_+(c)$;
12 **for** each edge e covered by $KNLC(c)$ **do**
13 update $Upp(e)$ by any changed contribution of c;
14 sort each edge $e \in E$ by its upper bound $Upp(e)$;
15 ScanEdges(X, X_{s_i});
16 **if** X is updated **do**
17 $s' \leftarrow s, \mathcal{P} \leftarrow \{p | p$ is the optimal location with $X\}$;
18 **return** $\mathcal{P}$ and X and s';

Since $|C||V|\log|V|$ is typically the dominating factor, the time complexity is essentially k times that of the *MAS* algorithm.

6 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 website[1)]. 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.

6.1 Experiments for KMaxSum querying

Firstly, we compare the baseline algorithm in Section 2 with the *MAS* algorithm in Section 3 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 Fig. 3 and Fig. 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

[1)] http://www.dis.uniroma1.it/challenge9/download.shtml

$|S|$ 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 Fig. 3(a) and Fig. 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 Fig. 3(b) and Fig. 4(b), when $|S|$ is large, the increase of memory consumption is very small with $|S|$.

Effect of the number $|C|$ of clients and k: The results on *SF* and *COL* are in Fig. 5 and Fig. 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.

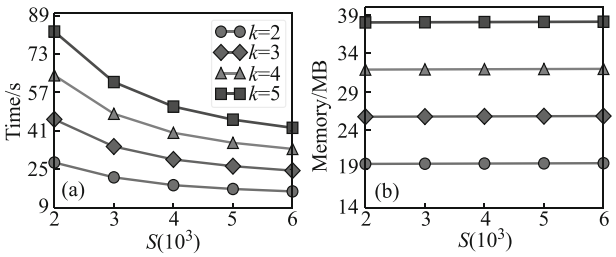

Fig. 3 Effect of $|S|$ and k on *SF* for *MAS*. (a) Time; (b) storage

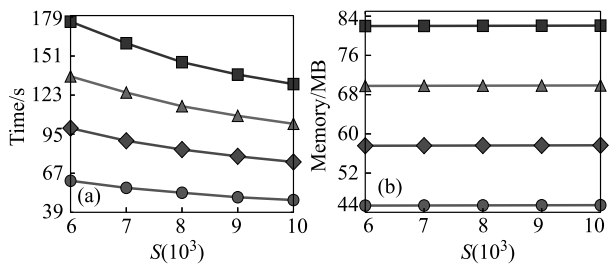

Fig. 4 Effect of $|S|$ and k on *COL* for *MAS*. (a) Time; (b) storage

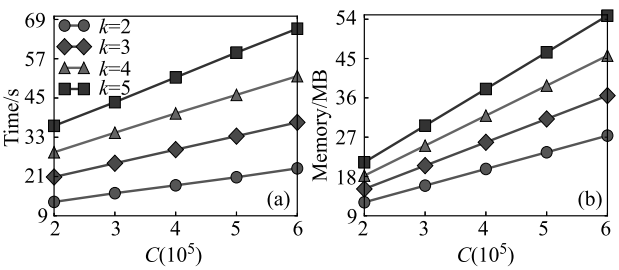

Fig. 5 Effect of $|C|$ and k on *SF* for *MAS*. (a) Time; (b) storage

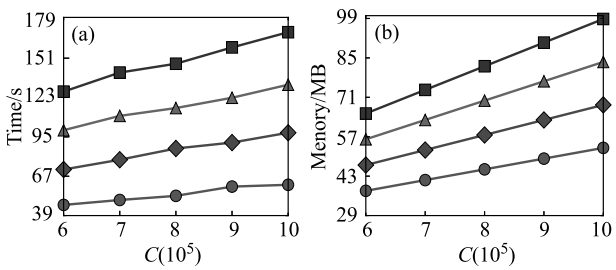

Fig. 6 Effect of $|C|$ and k on *COL* for *MAS*. (a) Time; (b) storage

Table 2 Effect of k and $|L|$ on *SF* for *MAS*

SF	Time/s			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

Table 3 Effect of k and $|L|$ on *COL* for *MAS*

COL	Time/s			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

Table 4 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 5 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

Effect of k and the number $|L|$ of class labels: The results on *SF* and *COL* are in Table 2 and Table 3, 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.

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 4, 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 5 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 Fig. 7 and Fig. 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 shown in these figures, both *Upp* and *NewUpp* have similar memory consumption. The lines for

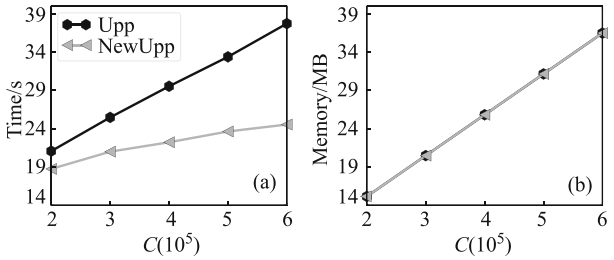

Fig. 7 Effect of $|C|$ on *SF* for *Upp* and *NewUpp*. (a) Time; (b) storage

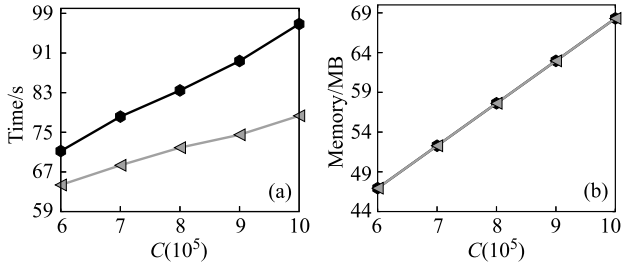

Fig. 8 Effect of $|C|$ on *COL* for *Upp* and *NewUpp*. (a) Time; (b) storage

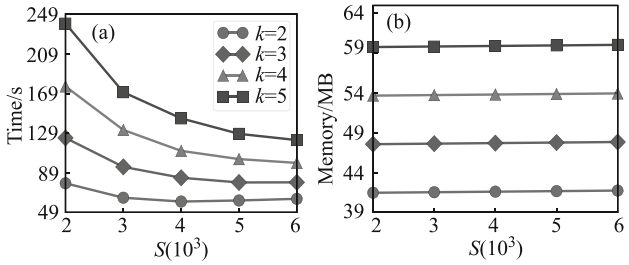

Fig. 9 Effect of $|S|$ and k on *SF* for *RMAS*. (a) Time; (b) storage

memory consumption are overlapping in these figures.

Since the OLQ with the objective MaxSum can be viewed as a special case of the KMaxSum query, we also compare the *MAS* algorithm with the state-of-the-art method called *MaxSum-Alg* in [10, 11] for the OLQ. With the default parameter values, the algorithm *MaxSum-Alg*/*MAS* requires 3.51/8.21 seconds and 10.03/13.62 MB on *SF*, and 13.3/24.73 seconds and 25.81/33.21 MB on *COL*. Thus, the *MaxSum-Alg* requires a little less time and memory storage than the *MAS*. But, the *MaxSum-Alg* can only handle the case of KOLQ where $k = 1$.

6.2 Experiments for Relocation KMaxSum querying

Effect of the number $|S|$ of servers and k: In general, the time of *RMAS* depends on three factors. Two factors are the same as the factors discussed for *MAS*. The additional factor is the processing of the potential servers to be moved. When $|S|$ is larger, the number of potential servers to be moved is larger and thus

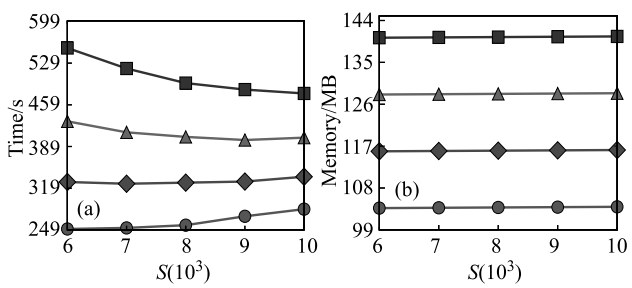

Fig. 10 Effect of $|S|$ and k on *COL* for *RMAS*. (a) Time; (b) storage

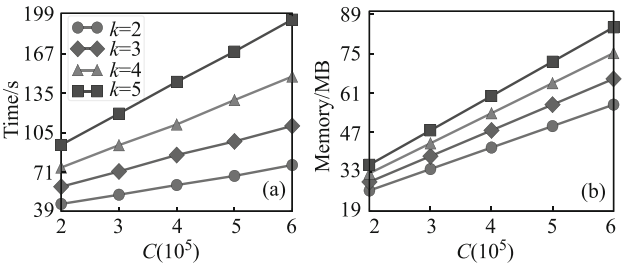

Fig. 11 Effect of $|C|$ and k on *SF* for *RMAS*. (a) Time; (b) storage

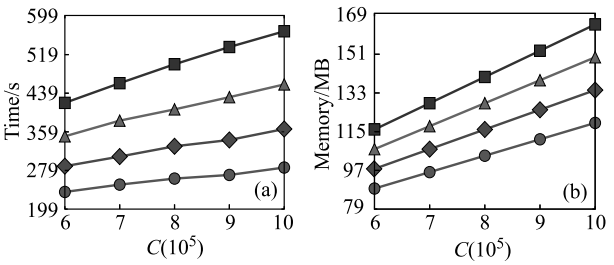

Fig. 12 Effect of $|C|$ and k on *COL* for *RMAS*. (a) Time; (b) storage

the time of *RMAS* may increase. However, the radius of KNLCs of clients may be reduced with the increase of $|S|$ and thus the time of *RMAS* becomes smaller. Based on these factors, the time of *RMAS* fluctuates with $|S|$ as shown in Fig. 9 and Fig. 10. The reasons for the memory consumption of *RMAS* are similar to *MAS*.

Effect of the number $|C|$ of clients and k: The results on *SF* and *COL* are shown in Fig. 11 and Fig. 12, respectively. The time and the memory consumption of *RMAS* increase with increases of $|C|$ and k. Similar reasons can be found for *MAS* with the effect of $|C|$ and k, and it is also consistent to our analysis of its time complexity.

7 Conclusion

In this paper, we study the KOLQ problem with the KMaxSum objective function. We also study a variant of KOLQ for relocation of servers on a road network, which we call RKOLQ. We propose two location analysis algorithms for the problem and its variant. 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 location analysis with reasonable time and memory. The KOLQ problem with other objective functions will be studied in our future work.

Acknowledgements We would like to thank all the anonymous reviewers for their insightful and helpful comments. This paper was supported by the National Nature Science Foundation of China (Grant Nos. 61572537, U1501252).

References

1. Cabello S, Diaz-Banez J M, Langerman S, Seara C, Ventura I. Reverse Facility Location Problems. Ljubljana: University of Ljubljana, Department of Mathematics, 2006
2. Cardinal J, Langerman S. Min-max-min geometric facility location problems. In: Proceedings of European Workshop on Computational Geometry. 2006, 149–152
3. Jakob K R A, Pruzan P M. The simple plant location problem: survey and synthesis. European Journal of Operational Research, 1983, 12: 36–81

4. Tansel B C, Francis R L, Lowe T J. Location on networks: a survey. Management Science, 1983, 29(4): 498–511
5. Xiao X K, Yao B, Li F F. Optimal location queries in road network databases. In: Proceedings of IEEE International Conference on Data Engineering. 2011, 804–815
6. Van Kreveld M, Schwarzkopf O, de Berg M, Overmars M. Computational Geometry Algorithms and Applications. Heidelberg: Springer, 2000
7. Yao B, Xiao X K, Li F F, Wu Y F. Dynamic monitoring of optimal locations in road network databases. The VLDB Journal—The International Journal on Very Large Data Bases, 2014, 23(5): 697–720
8. Qi J Z, Zhang R, Kulik L, Lin D, Xue Y. The min-dist location selection query. In: Proceedings of IEEE International Conference on Data Engineering. 2012, 366–377
9. Qi J Z, Zhang R, Wang Y Q, Xue A Y, Yu G, Kulik L. The min-dist location selection and facility replacement queries. World Wide Web, 2014, 17(6): 1261–1293
10. Chen Z, Liu Y B, Wong R C W, Xiong J M, Mai G L, Long C. Efficient algorithms for optimal location queries in road networks. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2014, 123–134
11. Chen Z T, Liu Y B, Wong R C W, Xiong J M, Mai G L, Long C. Optimal location queries in road networks. ACM Transactions on Database Systems, 2015, 40(3): 17
12. Gu Y, Zhang H, Wang Z, Yu G. Efficient moving k nearest neighbor queries over line segment objects. World Wide Web, 2016, 19(4): 653–677
13. Cui J T, Wang M, Li H, Cai Y. Place your next branch with MILE-RUN: min-dist location selection over user movement. Information Sciences, 2018, 463: 1–20
14. Shokeen S. A study on fast food consumption pattern in India. International Journal of Research and Engineering, 2016, 3(12): 10–15
15. Chen Z T, Liu Y B, Fu A W C, Wong R C W, Dai G N. KOLQ in a road network. In: Proceedings of IEEE International Conference on Mobile Data Management. 2019, 81–90
16. Sankaranarayanan J, Samet H, Alborzi H. Path oracles for spatial networks. Proceedings of the VLDB Endowment, 2009, 2(1): 1210–1221
17. Hu H, Lee D L, Lee V. Distance indexing on road networks. In: Proceedings of the International Conference on Very Large Data Bases. 2006, 894–905
18. Samet H, Sankaranarayanan J, Alborzi H. Scalable network distance browsing in spatial databases. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. 2008, 43–54
19. Mouratidis K, Yiu M L, Papadias D, Mamoulis N. Continuous nearest neighbor monitoring in road networks. In: Proceedings of the 32nd International Conference on Very Large Data Bases. 2006, 43–54
20. Cheema M A, Zhang W, Lin X, Zhang Y, Li X F. Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. The VLDB Journal—The International Journal on Very Large Data Bases, 2012, 21(1): 69–95
21. Yan D, Zhao Z, Ng W. Efficient algorithms for finding optimal meeting point on road networks. Proceedings of the VLDB Endowment, 2011, 4(11): 968–979
22. Qi J Z, Xu Z H, Xue Y, Wen Z Y. A branch and bound method for min-dist location selection queries. In: Proceedings of the 23rd Australasian Database Conference–Volume 124. 2012, 51–60
23. Huang J, Wen Z Y, Qi J Z, Zhang R, Chen J, He Z. Top-k most influential locations selection. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management. 2011, 2377–2380
24. Gao Y J, Qi S Y, Chen L, Zheng B H, Li X H. On efficient k-optimal-location-selection query processing in metric spaces. Information Sciences, 2015, 298: 98–117
25. Wong R C W, Ozsu M T, Fu A W C, Yu P S, Liu L. Efficient method for maximizing bichromatic reverse nearest neighbor. Proceedings of the VLDB Endowment, 2009, 2(1): 1126–1137
26. Wong R C W, Ozsu M T, Fu A W C, Yu P S, Liu L, Liu Y B. Maximizing bichromatic reverse nearest neighbor for Lp-norm in two-and three-dimensional spaces. The VLDB Journal—The International Journal on Very Large Data Bases, 2011, 20(6): 893–919
27. Yan D, Wong R C W, Ng W. Efficient methods for finding influential locations with adaptive grids. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management. 2011, 1475–1484
28. Liu Y B, Wong R C W, Wang K, Li Z J, Chen C, Chen Z T. A new approach for maximizing bichromatic reverse nearest neighbor search. Knowledge and Information Systems, 2013, 36(1): 23–58
29. Zhou Z N, Wu W, Li X H, Li M L, Hsu W. Maxfirst for maxbrknn. In: Proceedings of IEEE International Conference on Data Engineering. 2011, 828–839
30. Liu R F, Fu A W C, Chen Z T, Huang Sl L, Liu Y B. Finding multiple new optimal locations in a road network. In: Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2016, 1–10
31. Ghaemi P, Shahabi K, Wilson J P, Kashani F B. A comparative study of two approaches for supporting optimal network location queries. GeoInformatica, 2014, 18(2): 229–251
32. Gamper J, Bohlen M, Innerebner M. Scalable computation of isochrones with network expiration. In: Proceedings of International Conference on Scientific and Statistical Database Management. 2012, 526–543
33. Xu Z D, Jacobsen H A. Processing proximity relations in road networks. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2010, 243–254
34. Yilmaz E, Elbasi S, Ferhatosmanoglu H. Predicting optimal facility location without customer locations. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2017, 2121–2130
35. Du Y, Zhang D H, Xia T. The optimal-location query. In: Proceedings of International Symposium on Spatial and Temporal Databases. 2005, 163–180
36. Zhang D H, Du Y, Xia T, Tao Y F. Progressive computation of the min-dist optimal-location query. In: Proceedings of the International Conference on Very Large Data Bases. 2006, 643–654
37. Yiu M L, Mamoulis N, Papadias D. Aggregate nearest neighbor queries in road networks. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(6): 820–833
38. Papadias D, Tao Y F, Mouratidis K, Hui K. Aggregate nearest neighbor queries in spatial databases. ACM Transactions on Database Systems, 2005, 30(2): 529–576
39. Elmongui H G, Mokbel M F, Aref W G. Continuous aggregate nearest neighbor queries. GeoInformatica, 2013, 17(1): 63–95
40. Korn F, Muthukrishnan S. Influence sets based on reverse nearest neighbor queries. ACM Sigmod Record, 2000, 29(2): 201–212
41. Vlachou A, Doulkeridis C, Kotidis Y, Norvag K. Monochromatic and bichromatic reverse top-k queries. IEEE Transactions on Knowledge and Data Engineering, 2011, 23(8): 1215–1229
42. Bernecker T, Emrich T, Kriegel H P, Renz M, Zankl S, Zufle A. Efficient probabilistic reverse nearest neighbor query processing on uncertain data. Proceedings of the VLDB Endowment, 2011, 4(10): 669–680
43. Cheema M A, Lin X, Wang W, Zhang W J, Pei J. Probabilistic reverse nearest neighbor queries on uncertain data. IEEE Transactions on Knowledge and Data Engineering, 2009, 22(4): 550–564
44. Xu L, Mai G L, Chen Z T, Liu Y B, Dai G N. MinSum based optimal location query in road networks. In: Proceedings of International Conference on Database Systems for Advanced Applications. 2017, 441–457
45. Dijkstra E W. A note on two problems in connexion with graphs. Numerische Mathematik, 1959, 1(1): 269–271

Yubao Liu is currently a professor with the Department of Computer Science of Sun Yat-Sen University, China. He received his PhD in computer science from Huazhong University of Science and Technology, China in 2003. He has published more than 50 refereed journal and conference papers including SIGMOD, TODS, VLDB and VLDBJ, etc. His research interests include database systems and data mining. He is a senior member of the China Computer Federation (CCF).

Zitong Chen is currently working towards the PhD degree in the Department of Computer Science and Engineering, the Chinese University of Hong Kong, China. His research interests include database, data mining, and machine learning.

Ada Wai-Chee Fu received her BSc degree in computer science in the Chinese University of Hong Kong, China in 1983, and both MSc and PhD degrees in Computer Science in Simon Fraser University of Canada in 1986, 1990, respectively; worked at Bell Northern Research in Ottawa, Canada from 1989 to 1993; joined the Chinese University of Hong Kong, China in 1993.

Raymond Chi-Wing Wong is an associate professor in Computer Science and Engineering (CSE) of The Hong Kong University of Science and Technology (HKUST), China. He is currently the associate director of the Data Science and Technology (DSCT) program. He was the director of the Risk Management and Business Intelligence (RMBI) program from 2017 to 2019, the director of the Computer Engineering (CPEG) program from 2014 to 2016 and was the associate director of the Computer Engineering (CPEG) program from 2012 to 2014. He received the BSc, MPhil and PhD degrees in Computer Science and Engineering in the Chinese University of Hong Kong (CUHK), China in 2002, 2004 and 2008, respectively.

Genan Dai is currently working towards the PhD degree in the School of Data and Computer Science, Sun Yat-Sen University, China. Her research interests include data mining and artificial intelligence.