remote sensing

MDPI

Article

In Search of the Max Coverage Region in Road Networks

Lanting Fang [1], Ze Kou [1], Yuzhang Zhou [1], Yudong Zhang [2,*] and George Y. Yuan [3]

[1] School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China
[2] School of Computing and Mathematical Sciences, University of Leicester, Leicester LE1 7RH, UK
[3] Thinvent Digital Technology Co., Ltd., Nanchang 330096, China
* Correspondence: yudong.zhang@le.ac.uk

Abstract: The widespread use of mobile devices has resulted in the generation of vast amounts of spatial data. The availability of such large-scale spatial data facilitates the development of data-driven approaches to address real-life problems. This paper introduces the max coverage region (MCR) problem in road networks and provides efficient solutions. Given a set of spatial objects and a coverage radius, the MCR problem aims to identify a location from the road network, so that we can reach as many spatial objects as possible within the given coverage radius from the location. This problem is fundamental to supporting many real-world applications. Given a road network and a set of sensors, this problem can be used to find the best location for a sensor maintenance station. This problem can also be applied in medical research, such as in a protein–protein interaction network, where the nodes represent proteins, the edges represent their interactions, and the weight of an edge represents confidence. We can use the MCR problem to find the set of interacting proteins with a confidence budget. We propose an efficient exact solution to solve the problem, where we reduce the MCR problem to an equivalent problem named the most overlapped interval and design an edge-level upper bound estimation method to reduce the search space. Furthermore, we propose two approximate solutions that sacrifice a little accuracy for much better efficiency. Our experimental study on real-road network datasets demonstrates the effectiveness and superiority of the proposed approaches.

Keywords: road network; region search; location selection; spatial data management; shortest distance

Citation: Fang, L.; Kou, Z.; Zhou, Y.; Zhang, Y.; Yuan, G.Y. In Search of the Max Coverage Region in Road Networks. *Remote Sens.* **2023**, *15*, 1289. https://doi.org/10.3390/rs15051289

Academic Editor: Maria Antonia Brovelli

Received: 25 January 2023
Revised: 22 February 2023
Accepted: 23 February 2023
Published: 26 February 2023

1. Introduction

With the proliferation of GPS-enabled mobile devices, users can access various location-based services, such as Foursquare and Google Maps. Therefore, massive volumes of spatial data are being generated rapidly every day. The availability of such large-scale spatial data facilitates the development of data-driven approaches to address problems in real scenarios, such as urban planning. Consider the following example.

Example 1. *Suppose that the government has already placed a set of sensors along the road network. In order to collect data from the sensors and guarantee the sensors work properly, the government would desire to set up a maintenance station to host engineers. Intuitively, a good location for the station should reach as many sensors as possible within a reasonable distance. Where should the station be set up?*

In this paper, we generalize the motivating example and formulate the *max coverage region* (MCR) problem. Informally, given a set of spatial objects in the road network, we wish to find a location, such that the objects within its coverage are maximized. Here, we assume that an object is in the coverage of a location if the road network distance between the object and the location is no larger than a given threshold.

Example 2. *Consider the example in Figure 1. The road network consists of nine nodes and nine edges. The black dots represent four spatial objects. Assume the coverage radius is 4. When the station is placed at the black square, it covers o_1, o_2, and o_3. The shortest path between the station and each covered object is highlighted in red. As no other location can cover more than three spatial objects, the black square is the result that the MCR problem aims to return.*

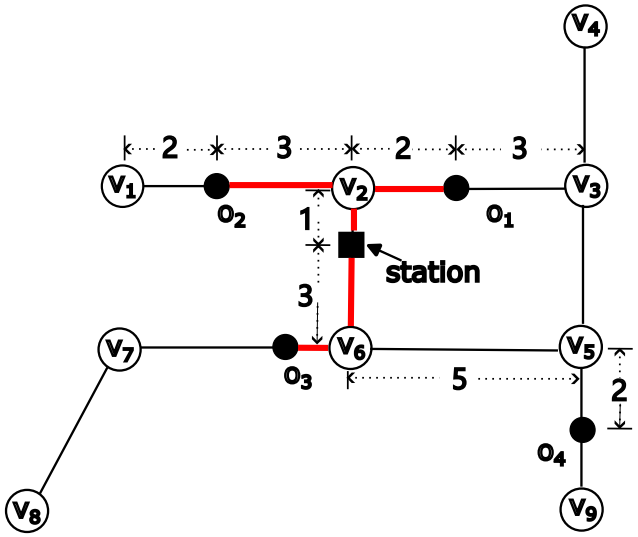


Figure 1. An example of a road network. The numbers on the edge indicates the length of the road segments.

It is non-trivial to solve the MCR problem. In the road network, there are infinite candidate locations and it is prohibitively expensive to consider all these locations in the search of the max coverage region. Moreover, in order to determine whether the region at a location has the maximum coverage, it is required to compute the network distance between the location and many objects, which is also expensive. Thus, these demand techniques that can efficiently identify the optimal location over very large volumes of spatial objects.

To address the aforementioned challenges, we first propose the *most overlapped interval* (MOI) problem. Instead of searching for the optimal location from the infinite set of candidate locations, we construct an equivalent *MOI* problem and design an effective algorithm based on a linear scan to solve it. This reduction greatly reduces the search space. However, even if we can search an edge efficiently, it becomes prohibitively expensive if we search every edge. Therefore, we propose an edge-level upper-bound estimation method. By acquiring the upper bounds for every edge, we search the edges in a greedy manner and terminate immediately when the upper bound of the remaining edges is worse than the current known optimal location. In some scenarios, it is acceptable to sacrifice a little accuracy for much better efficiency. Motivated by this, we further propose two highly efficient solutions that find results with comparable quality. According to our experiments, the runtimes of the approximate solutions are only $\frac{1}{8}$ of the exact solutions. To demonstrate the superiority of our proposed solutions, we conducted extensive experiments on five real-world road network datasets. The experiments show that our solutions are capable of handling large-scale datasets and could find satisfying results.

Our paper is related to the *community search/detection* problem, the *best region search* problem, and the *location selection* problem. We will elaborate on these three problems in turn.

Best region search problem. The best region search problem aims to find the location of a given-size rectangular region such that the score of the region is maximized. This kind of problem is first proposed and investigated as the *maximizing range sum* (MaxRS) problem [1,2] in the computational geometry community. The score of a region in the MaxRS problem is the number of spatial objects inside the region. Imai et al. [2] developed

an efficient algorithm to find the position of the rectangle of the given size enclosing the maximum number of spatial objects. The algorithm's complexity is $O(n \log n)$, where n is the number of spatial objects. Nandy and Bhattacharya [1] reduced the MaxRS problem to a rectangle overlapping problem and then designed a sweep-line-based algorithm with the same time complexity $O(n \log n)$. Choi et al. [3] thoroughly investigated the MaxRS problem. They assumed that the scalability of the spatial object was too large to be placed in memory. They proposed an external memory algorithm based on the in-memory $O(n \log n)$ algorithm. Tao et al. [4] proposed sacrificing some accuracy for better efficiency. They proposed a $(1 - \epsilon)$-approximate algorithm for addressing the MaxRS problem efficiently. Feng et al. [5] generalized the MaxRS problem to propose the best region search problem, where the score of a region was generalized from the SUM to a user-defined submodular monotone function. Mostafiz et al. [6] extended the MaxRS problem by taking the types of spatial objects into account. Users can issue constraints on the types of spatial objects while searching for the region with the maximum total weight. Feng et al. [7] next proposed an attribute-aware similar function to evaluate the similarity between regions. Users can issue a query region to search for regions that have similar attribute distributions. Moreover, researchers proposed the use of a semantic window [8] and searchlight [9] to investigate the region search problem in an interactive data exploration manner for multidimensional data. All of this work focuses on finding the location of a rectangular region in the Euclidean space. In this paper, we aim to find the optimal location for a station whose coverage is defined as a subgraph on the road network. In addition, the aforementioned region search problems require users to specify the width and the height of the rectangular region, which is difficult to set. In contrast, in this paper, we eliminate the shape-related parameter and only let the users specify the radius of the coverage, which is intuitive and straightforward. Thus, the aforementioned studies are different from our problem.

The *length-constrained maximum-sum region* (LCMSR) query [10] is the most relevant to our work. Given a spatial network and a set of query keywords, the LCMSR query returns a subgraph from the road network such that (1) the distance between any pair of nodes in the subgraph is no larger than a threshold, and (2) the total textual relevance between the keywords on each node in the subgraph and the query keywords is maximized. The LCMSR problem differs from our problem in the following aspects: (1) The LCMSR problem imposes a length constraint on the pairwise distance to control the shape of the region, making the LCMSR problem NP-hard. In contrast, in our paper, we assume the coverage of a station is a subgraph centered at a location in the road network. As we will show in Section 2.2, our proposed problem can be solved in polynomial time. (2) The LCMSR problem only considers the nodes in the spatial network; however, in this paper, we assume that spatial objects can be located at any location on any edge in the road network. Therefore, the techniques proposed for LCMSR cannot be applied to solve our problem.

Location selection problem. Our MCR problem is related to the *location selection* problem. The *location selection* problem aims to find the location for a new facility, such that a certain objective function is optimized. As input, the *min-dist optimal-location* query [11] uses a set S of sites, a set O of weighted objects, and a spatial region Q. It returns a location in Q, which, if a new site is built there, minimizes the average distance from each object to its closest site. Xiao et al. [12] extended the location selection problem to road networks, as movements between spatial locations are usually confined by the underlying road network in practice. Chen et al. [13] also investigated the location selection problem in road networks, but they considered minimizing the maximum distance between any client and the site that served him/her. Another class of studies aimed to find the location for a new server to maximize its influence. For instance, Du et al. [14] assumed that the influence of a new site is the total weight of its reverse nearest neighbors, i.e., the total weight of objects that are closer to this site than the others. They consider this problem in the L1-norm space. Moreover, Cabello et al. [15] studied this problem in the L2-norm space and a very efficient algorithm was proposed [16]. Choudhury et al. [17] incorporated textual information into this problem and only considered the reverse k-nearest neighbors that were textual relevant

to the query keywords. Zhou et al. [18] and Yan et al. [19] also considered the problem with different definitions of influence, respectively. Moreover, many works [20–22] investigated the problem of selecting the top-k locations. The location selection problem differs from the MCR problem, as it cares about the reverse nearest neighbors of the new site. In other words, the new site has to compete with other sites to affect more objects. In our paper, the location of the station is independent. We only need to find the optimal location to cover as many objects as possible without considering the effect of other competing stations.

The contributions of this paper are summarized as follows:

- We formulated the *max coverage region* (MCR) problem. To reduce the search space, we propose a *most overlapped interval* (MOI) problem and prove that it is equivalent to solving the *MOI* problem. This idea greatly reduces the search space from infinite candidate locations to $O(|S|)$ intervals.
- To address the MCR problem, we developed an edge-level upper-bound estimation method. This enabled us to search the road network in a greedy manner.
- We propose two approximate solutions. They are highly efficient and return results with comparable quality.
- Extensive experiments were conducted on five real-road network datasets. The results demonstrate the efficiency and effectiveness of our proposed solutions.

The remainder of this paper is organized as follows. Section 2 formally introduces the max coverage region problem and presents the details of the solutions. Section 3 presents the results and the discussions. Finally, Section 4 concludes the paper.

2. Method

2.1. Problem Definition

In this subsection, we first present the max coverage region (MCR) problem.

Definition 1 (Road Network Graph). *A road network $G = (V, E, w)$ consists of a set of nodes V, a set of undirected edges $E \subseteq V \times V$, and a distance function $w : E \to \mathbb{R}$. Each node $v \in V$ is a spatial location $v = (lat, lng)$, where (lat, lng) are its coordinates.*

A set O of data objects resides on the road network. Each data object $o_i \in O$ is a point on an edge $e \in E$.

The road network graph models the roads in real life, where each node $v \in V$ represents a road junction, and each edge $e \in E$ represents a road segment. Data objects can be the locations of vehicles or pedestrians.

Consider the example in Figure 1, where the road network consists of nine nodes and nine edges. There are four data objects o_1, o_2, o_3, and o_4. The four data objects are located on edges (v_2, v_3), (v_1, v_2), (v_6, v_7) and (v_5, v_9), respectively.

Remark 1. *Note that in this paper, we assume that the spatial objects are located on the edges of the road network. At first glance, it may seem that we can treat the spatial objects as nodes and build a new graph. However, this is not appropriate due to the following reasons: (1) In real-life scenarios, the spatial objects are usually used to represent points of interest, user check-ins, and GPS locations. Compared with the road network, such spatial objects may change rapidly. If we transform these spatial objects in the nodes in the network, we may need to frequently update the network when these spatial objects change, which would be very inefficient. (2) As we treat the spatial objects separately, we can store and manage the set of spatial objects on each edge together and build an efficient index. This enables us to access spatial objects efficiently.*

Definition 2 (Coverage). *Let p be a location in the road network. Given a threshold γ, a data object (o) is in p's coverage if the road network distance between p and o is no larger than γ, i.e., $dist(p, o) \leq \gamma$.*

In Figure 1, the shortest paths between the station and o_1, o_2, and o_3 are highlighted in red. In Figure 1, the distances between the station and o_1, o_2, and o_3 are 3, 4.

Definition 3 (The MCR Problem)**.** *Consider a road network graph G. Given a set (O) of data objects that reside on the road network and a threshold γ, and given a road network graph and a set of data objects that reside on the road network, the MCR problem aims to find the optimal location for a station, such that the number of data objects that are covered by the station is maximized.*

2.2. An Exact Solution

The MCR problem requires selecting the best location for a station to cover as much as spatial objects as possible. Intuitively, the optimal location can be located at any point in the road network. Apparently, it is prohibitively expensive to enumerate all points, which is infinite. To tackle this challenge, we first present an effective exact method for the MCR problem. To this end, we first introduce how to select the best point on a given edge (u, v) that covers the maximum number of spatial objects.

2.2.1. Searching an Edge for the Optimal Location

To search for the point that covers the maximum number of spatial objects of an edge, a naive idea is to evaluate the number of objects that are covered by each point on this edge. For instance, take Figure 2a as an example, where the blue point p is a point on the edge (v_1, v_4). For the blue point p on the edge (v_1, v_4), we explore close objects within the given radius by expanding from p. The coverage of p is highlighted in red. As can be seen from the figure, objects o_1 and o_3 are covered by p. By repeating this procedure for every point on the edge (v_1, v_4), we can find the optimal location on this edge. Unfortunately, as there is an infinite number of points, this idea is prohibitively expensive.

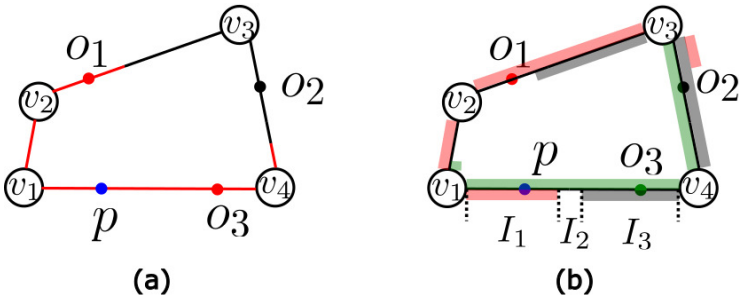


Figure 2. In (**a**), the coverage of point p is highlighted in red. Point p covers spatial objects o_1 and o_3. In (**b**), the coverage of o_1, o_2 and o_3 are highlighted in red, grey and green, respectively. Point p is in the coverage of o_1 and o_3.

To address the above issue, we propose the *most overlapped interval* (MOI) problem. We then show that we can reduce the problem of searching for the optimal point on the edge to solve the most overlapped interval problem, which can be solved efficiently. We next give the formal definition of the *MOI* problem and illustrate how to perform the reduction with a detailed example.

Definition 4 (Most Overlapped Interval)**.** *Given an edge (u_1, u_2), a segment is a part of the edge bounded by two endpoints p^s and p^e on the edge. Let $S = \{(p_1^s, p_1^e), \ldots, (p_k^s, p_k^e)\}$ be a set of segments on the edge. The most overlapped interval problem aims to find the interval (p^s, p^e), $p^s, p^e \in \{p_1^s, \ldots, p_k^s, p_1^e, \ldots, p_k^e\}$, such that (p^s, p^e) overlaps with the maximum number of segments.*

Example 3. *Consider the example in Figure 3, where there is one edge and four segments. The endpoints of the segments divide the edge into six intervals. We notice that intervals I_1 and I_2 overlap with three segments, which is more than other intervals. Thus, both I_1 and I_2 are the most overlapped intervals on the edge.*

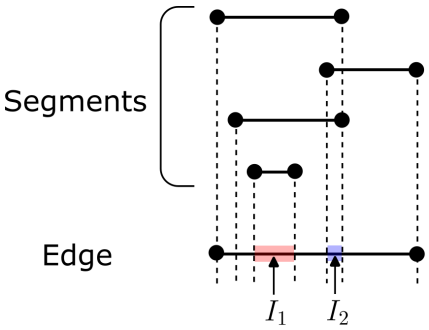


Figure 3. Most overlapped interval problem.

To explain how we can reduce the problem of searching for the optimal point on the edge of the *MOI* problem, we first present the following observation. Consider edge (v_1, v_4) in the example in Figure 2a. We observe that spatial object o_1 is in the coverage of point p (highlighted in red). Inversely, if we draw the coverage of o_1, we notice that point p is located at the interval I_1, which is the overlap between o_1's coverage and the edge (v_1, v_4). In fact, any point on the interval I_1 can cover o_1. Motivated by this observation, a natural idea to identify the overlap between (v_1, v_4) and every spatial object's coverage. Each overlap corresponds to a segment on the edge (v_1, v_4). We can then find the most overlapped interval. Any point on this interval is an optimal location on the edge (v_1, v_4) that covers the maximum number of spatial objects. The following example demonstrates the reduction from Figure 2a.

Example 4. *There are three spatial objects (o_1, o_2, and o_3) in Figure 2a. For each spatial object, we draw the coverage of o_1, o_2, and o_3, which are highlighted in red, gray, and green, respectively. Each object's coverage overlaps with edge (v_1, v_4) and the resulting segments divide the edge into three different intervals, denoted as I_1, I_2, and I_3. Interval I_1 overlaps with the red and the green segments, while I_3 overlaps with the green and the gray segments. Thus, I_1 and I_3 are the most overlapped intervals. We can pick a point p from I_1 and return it to users as a result.*

We showed how to reduce the problem of searching an edge for the optimal point to the most overlapped interval problem. Two questions are to be answered: (1) Which spatial objects should we consider to compute the overlap between the edge and their coverage so that we can generate the segments for the *MOI* problem? (2) How do we address the *MOI* problem efficiently? We next elaborate on the two questions in turn.

Identifying valid spatial objects.

As we demonstrated above, the input of the reduced *MOI* problem is the set of segments, each of which is the overlap between a spatial object's coverage and the edge. Some spatial objects are very far from the edge, such that any point on the edge cannot reach the object. It is unnecessary to consider such spatial objects while we are generating the overlapped segments for the *MOI* problem. Thus, to identify the valid spatial objects to generate segments for the reduced *MOI* problem on the edge (u, v), we can divide the spatial objects into three classes as follows:

Case 1: Irrelevant objects. A spatial object (o) is an irrelevant object to the edge (u, v) if no point on (u, v) can reach o within the radius γ. As a result, we do not need to consider such objects when searching for the most overlapped interval. We can easily derive the following lemma to verify whether a spatial object is irrelevant.

Lemma 1. *If $dist(o, u) > \gamma$ and $dist(o, v) > \gamma$, then o is an irrelevant object to (u, v).*

Proof. For any point p on (u, v), its distance to o is $dist(p, o) = \min(dist(p, u) + dist(u, o), dist(p, v) + dist(v, o))$. Since $dist(u, o) > \gamma$, $dist(v, o) > \gamma$, it is obvious that $dist(p, o) > \gamma$. Thus, o is an irrelevant object. □

Case 2: Fully-covered objects. A spatial object (o) is a fully-covered object to edge (u, v) if any point on (u, v) can reach o within the radius γ. Thus, it is unnecessary to consider o anymore when searching the edge (u, v) for the most overlapped interval. We derive the following lemma to verify whether o is a fully-covered object.

Lemma 2. *An object (o) is a fully-covered object to (u, v) if at least one of the following conditions holds:*

1. $dist(o, u) < \gamma - |(u, v)|$;
2. $dist(o, v) < \gamma - |(u, v)|$;
3. $dist(o, u) + dist(o, v) <= 2\gamma - |(u, v)|$.

Proof. It is obvious that o is a fully-covered object when the first two conditions hold. For the third condition, consider a point p on the edge (u, v). We

$$\begin{aligned} dist(p, o) &= \min(dist(p, u) + dist(u, o), dist(p, v) + dist(v, o)) \\ &\leq \frac{1}{2} \cdot (dist(p, u) + dist(u, o) + dist(p, v) + dist(v, o)) \\ &= \frac{1}{2}(|(u, v)| + dist(u, o) + dist(v, o)) \end{aligned} \tag{1}$$

Since $dist(o, u) + dist(o, v) < 2\gamma - |(u, v)|$, we $dist(p, o) \leq \gamma$, i.e., any point (u, v) can reach o within the radius γ. □

Case 3: Partially covered objects. A spatial object (o) is a partially covered object to edge (u, v) if some points on (u, v) can reach o within the radius, while the other points cannot. Therefore, to search for the optimal point on the edge (u, v), we need to find all partially covered objects and determine the most influenced interval. To verify whether an object is a partially covered object to (u, v), we present the following lemma.

Lemma 3. *An object (o) is a partially covered object if the following two conditions hold:*

1. $dist(o, u) + dist(o, v) > 2\gamma - |(u, v)|$
2. $\gamma - |(u, v)| < dist(o, u) < \gamma$, *or* $\gamma - |(u, v)| < dist(o, v) < \gamma$

Proof. According to the definition of partially covered objects, we consider the following three cases: (1) As shown in Figure 4a, only the points on the interval (u, p_u^o) can reach o within γ through node u; (2) only the points on the interval (v, p_v^o) can reach o within γ through v; and (3) points on both intervals (u, p_u^o) and (v, p_v^o) can reach o within γ through u and v, respectively. By considering the three cases, we can easily derive the inequalities in the lemma. □

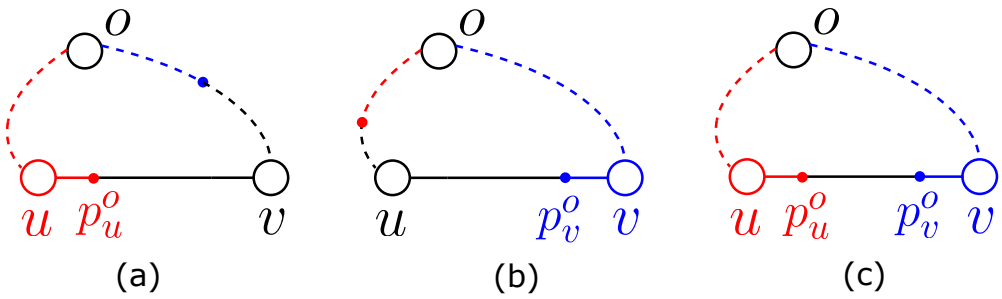


Figure 4. Three cases of partially covered objects. In (**a**), object o can be reached from edge (u, v) through u. In (**b**), object o can be reached from (u, v) through v. In (**c**), object o can be reached from (u, v) through either u and v.

The above three cases indicate that we only need to consider the segments concerning the intersection between the partially covered object's coverage and the edge (u, v).

Solve the *MOI* problem

We showed how to identify the set of spatial objects whose coverage on the edge (u, v) should be considered in the *MOI* problem. With the *MOI* problem constructed, how can one efficiently find the most overlapped interval? To answer this question, one straightforward idea is to maintain a counter for each interval and iterate over all segments to update the counter of every overlapped interval. We refer to this algorithm as the *MOIEnumerate* algorithm.

MOIEnumerate Algorithm. Algorithm 1 shows the pseudocode of the straightforward idea. The algorithm takes as input a set $\mathcal{S}$ of segments on the edge (u, v) and outputs the most overlapped interval I_{me}. The algorithm first collects the endpoints from all segments (lines 1–3) and sorts the endpoints (line 4). It computes all intervals bounded by every two consecutive endpoints (line 5). Next, the algorithm iterates over all segments (lines 6–8). For each segment, it computes the overlapped intervals (line 7) and increases the associated counters by 1 (line 8). When all segments are processed, we choose the interval with the largest counter (line 9) and return it to the user as the final result.

Algorithm 1: MOIEnumerate.

Input: A set S of segments on the edge (u, v)
Output: The most overlapped interval I_{mo} and the number of overlapping segments.

1 $B \leftarrow \emptyset$ **for** $(p^s, p^e) \in S$ **do**
2 $\quad B.add(p^s)$;
3 $\quad B.add(p^e)$
4 `Sort`(B);
5 $\mathcal{I} = \{(B[i], B[i+1])fori \in [0, |B| - 1]\}$;
6 **for** $s \in S$ **do**
7 $\quad \mathcal{I}_s \leftarrow$ the intervals that overlap with s **for** $I_i \in \mathcal{I}_s$ **do**
8 $\quad\quad I_i.count+ = 1$;
9 $I_{mo} \leftarrow \arg\max_{I \in \mathcal{I}} I.count$;
10 **return** I_{mo}, $I_{mo}.count$

Complexity Analysis. It takes $O(|S| \log |S|)$ to sort the endpoints of all segments, where $|S|$ is the number of segments. The $2|S|$ endpoints divide the edge into $O(S)$ intervals. Then, for each segment, we need to locate the intervals that are covered by the segment and update the counter for each interval. Thus, it takes $O(|S|)$ time to process one segment, and there are $O(|S|)$ segments to be processed. Therefore, the total complexity of Algorithm 1 is $O(|S|^2)$.

The *MOIEnumerate* algorithm is easy to implement. However, an expensive operation in the algorithm is the counter-updating operation, i.e., for each segment, we need to identify the overlapped intervals and update their counters in turn. The number of overlapping intervals is $O(|S|)$; thus, the *MOIEnumerate* algorithm scales poorly with respect to the number of segments.

MOILinearScan Algorithm. To tackle this efficiency issue of the *MOIEnumerate* algorithm, we next analyze the *MOI* problem and design an efficient linear scan-based algorithm, named *MOILinearScan*. We use the following example to explain the intuition behind the algorithm.

Example 5. *Consider the example in Figure 5. The endpoints of the segments divide the edge (u, v) into several disjoint intervals. In the example, $p_1, \ldots, p_8$ are the endpoints of four segments. We mark the left endpoints in red and the right endpoints in blue.*

Assume we move a sweep point from u to v (from left to right). The sweep point first meets the left endpoint p_1 of segment s_1. From now on, the sweep point overlaps with segment s_1. We keep moving the sweep point until it meets the left endpoint p_2 of segment s_2. The sweep point overlaps with s_1 and s_2 now. Similarly, when the sweep point meets the left endpoint p_3 of segment s_3, the sweep point overlaps with segments s_1, s_2, and s_3. Then, the next endpoint the sweep point meets is

p_4, which is the right endpoint of segment s_2. After that, the sweep point only overlaps with s_1 and s_3. We keep moving the sweep point until it reaches the end of this edge, i.e., v.

Assume a sweep point is moving from left to right. From the above example, we can observe the following properties:

Property 1. *When the sweep point meets a left endpoint of any segment, the next interval is overlapped with this segment.*

Property 2. *When the sweep point meets the right endpoint of any segment, the next interval is no longer overlapped with this segment.*

The two properties tell the status of consecutive intervals. Recall that each interval is bounded by two endpoints. If the endpoints do not overlap with each other, the difference between the number of overlapped segments of two consecutive intervals is 1. For instance, $\overline{p_1, p_2}$ overlap with one segment, while $\overline{p_2, p_3}$ overlap with two segments. Similarly, $\overline{p_3, p_4}$ overlap with three segments, while $\overline{p_4, p_5}$ overlap with two segments.

From the two properties, we derive the following lemma, which is the key to the *MOILinearScan* algorithm.

Lemma 4. *Among all intervals on the edge (u, v), the interval between a left endpoint and a right endpoint has the maximal number of overlapped segments, compared to its adjacent intervals.*

Proof. The two properties indicate the set of overlapped segments of such an interval and its adjacent intervals. Thus, the lemma is easily proved. □

This inspired us to use a design a sweep point-based method to search for the most overlapped interval. Specifically, we use a sweep point to scan the edge from left to right. During the sweeping, we maintain a counter to record the number of overlapped segments, keeping in mind the type of the recently passed endpoint, i.e., whether it is a left endpoint or a right endpoint of a segment. If we encounter an interval bounded by a left endpoint and a right endpoint consecutively, it is a maximal interval, which means the overlapped segments of this interval are larger than the adjacent intervals. We refer to this algorithm as *MOILinearScan*.

Algorithm 2 shows the pseudocode of *MOILinearScan*. The algorithm takes (as input) a set $\mathcal{S}$ of segments on the edge (u, v), and outputs the most overlapped interval I_{mo}. Initially, the algorithm collects the left and right endpoints of all segments and stores them in lists L and R, respectively (lines 2–3). Both lists are sorted in ascending order (line 4). The variable *start* is initialized as $L[0]$, representing the last endpoint the sweep point has encountered, and counter *count* (lines 5–6). Next, it iteratively pops endpoints from L and R, representing moving the sweep point from left to right (lines 7–15). In each iteration, if $L[0] < R[0]$, it means the sweep point meets a left endpoint, and the following interval's counter will be increased by 1 (lines 8–10). Otherwise, it means the sweep point meets the right endpoint and the current interval's counter is larger than its adjacent intervals. If the current interval's counter is larger than the record c_{max}, we record the interval and its counter with I_{mo} and c_{max}. When all left endpoints are processed, we can terminate the iteration (lines 16–17) and return I_{mo} to users.

Example 6. *Consider the example in Figure 5. The two lists are sorted as $L = [p_1, p_2, p_3, p_7]$, $R = [p_4, p_5, p_6, p_8]$. The variable start is initialized as p_1, and count is initialized as 1. In the first iteration, p_2 is 'popped' from L, as $p_2 = L[0] < R[0] = p_4$. We update start $= p_2$, count $= 2$. In the second iteration, p_3 is 'popped' from L, and start $= p_3$, count $= 3$. In the third iteration, p_4 is 'popped' from R. In this case, the interval $\overline{p_3, p_4}$ is a maximal interval. We use $c_{max} = 3$ and $I_{mo} = \overline{p_3, p_4}$ to record this interval. By repeating this process, we can know the counter for every interval. Since interval $\overline{p_3, p_4}$ has the maximum counter, we return it to the user as the final result.*

Complexity Analysis. It takes $O(|S| \log |S|)$ time to sort the left/right endpoints of the $|S|$ segments. We next process an endpoint from either L or R in every iteration, and there are $O(|S|)$ iterations. Putting these together, the total complexity of Algorithm 2 is $O(|S| \log |S|)$.

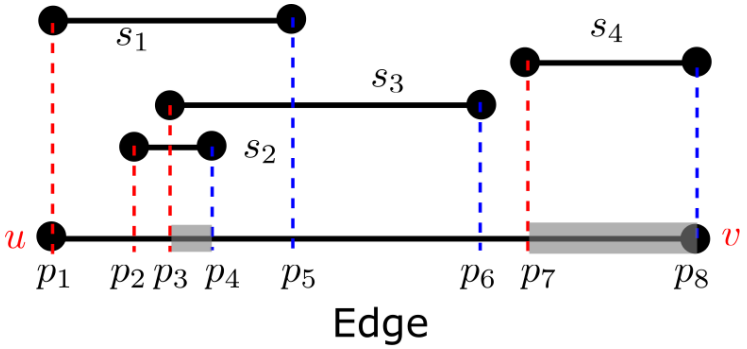


Figure 5. Property analysis of the *MOI* problem. The red dashed lines correspond to the left endpoints of intervals, while the blue dashed lines correspond to the right endpoints of intervals.

Algorithm 2: MOILinearScan.

Input: A set S of segments on the edge (u, v)
Output: The most overlapped interval I_{mo} and the number of overlapped segments.

$L \leftarrow \emptyset, R \leftarrow \emptyset, count = 0, c_{max} = -\infty$;
for $(p^s, p^e) \in S$ **do**
  $L.add(p^s), R.add(p^e)$
`Sort`(L), `Sort`(R);
$start \leftarrow L[0]$,$L.pop\_front()$;
$count \leftarrow 1$;
while $L \neq \emptyset$ and $R \neq \emptyset$ **do**
  if $L[0] < R[0]$ **then**
    $start \leftarrow L[0], count \leftarrow count + 1$;
    $L.pop\_front()$;
  else
    if $count > c_{max}$ **then**
      $c_{max} \leftarrow count, I_{mo} \leftarrow (start, R[0])$
    $count \leftarrow count - 1$;
    $start \leftarrow R[0], R.pop\_front()$;
  if L is $\emptyset$ **then**
    break;
return I_{mo}, c_{max}

2.2.2. Searching the Whole Road Network for the Optimal Location

In the previous subsection, we demonstrated how to search an edge for the optimal location, such that a station placed on that location covered the maximum number of spatial objects. Thus, in order to search the whole road network for the optimal point, the natural idea is to repeat the previous procedure on every edge of the road network. For instance, for every edge, we identify the partially covered objects and construct the corresponding *MOI* problem and invoke Algorithm 2 to solve it. Unfortunately, this idea involves many redundant operations, and it is unnecessary to examine every edge. In this subsection, we present a method to estimate an upper bound for every edge to show the maximum number of objects that a station on the edge can cover. With the upper bound, we can examine the edges in a greedy manner, which greatly reduces the search space.

Estimating Upper Bounds for Edges

The upper bound of an edge tells us what is the maximum number of covered objects if we place a station on the edge. A straightforward idea to define the upper bound is based on partially covered objects and fully-covered objects that we introduced in Section 2.2.1. Recall that fully-covered objects are the set of spatial objects that any point on the edge can reach within the radius, while partially covered objects are the set of spatial objects that only a part of points on the edge can reach within the radius. Therefore, The total number of fully-covered and partially covered objects is an upper bound for the maximum covered objects if we place a station on this edge. However, the spatial objects in the road network are usually very large and could emerge rapidly. It would be expensive to check every spatial object's status. In contrast, the road network is relatively small and seldom changes. As a result, we propose computing the upper bounds on the edge level.

To this end, a key question to be answered is: what is the upper bound of the distance between two points p_1 and p_2, where p_1 and p_2 are from two different edges? To answer this question, we use the example in Figure 6 to explain. In the example, (u_1, u_2) and (v_1, v_2) are two edges, and p_1 and p_2 are two points on the two edges, respectively. We can easily derive the distance between p_1 and p_2 as

$$\begin{aligned} dist(p_1, p_2) = \min(&dist(p_1, u_1) + dist(u_1, v_1), dist(v_1, p_2), \\ &dist(p_1, u_1) + dist(u_1, v_2), dist(v_2, p_2), \\ &dist(p_1, u_2) + dist(u_2, v_1) + dist(v_1, p_2), \\ &dist(p_1, u_2) + dist(u_2, v_2) + dist(v_2, p_2)) \end{aligned} \tag{2}$$

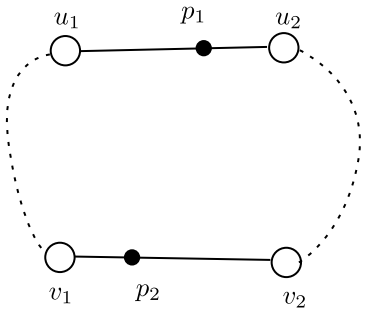


Figure 6. An example of analyzing the relationship between edges.

As it is obvious that $dist(p_1, u_1) \leq len(u_1, u_2)$, $dist(p_1, u_2) \leq len(u_1, u_2)$, $dist(p_2, v_1) \leq len(v_1, v_2)$, $dist(p_2, v_2) \leq len(v_1, v_2)$, where $len(v_1, v_2)/len(u_1, u_2)$ is the length of edge $(v_1, v_2)/(u_1, u_2)$. We define the function $UB((u_1, u_2), (v_1, v_2))$ as follows

$$\begin{aligned} UB((u_1, u_2), (v_1, v_2)) = \min\Big(&len(u_1, u_2) + dist(u_1, v_1) + len(v_1, v_2), \\ &len(u_1, u_2) + dist(u_1, v_2) + len(v_1, v_2), \\ &len(u_1, u_2) + dist(u_2, v_1) + len(v_1, v_2), \\ &len(u_1, u_2) + dist(u_2, v_2) + len(v_1, v_2), \\ &\frac{1}{4}\Big(2len(u_1, u_2) + 2len(v_1, v_2) + \sum_{u=\{u_1, u_2\}} \sum_{v=\{v_1, v_2\}} dist(u, v)\Big)\Big) \end{aligned} \tag{3}$$

We can easily derive that

$$dist(p_1, p_2) \leq UB((u_1, u_2), (v_1, v_2)) \tag{4}$$

The computation of function $UB(\cdot)$ only relies on the edge length ($len(u_1, u_2)$ and $len(v_1, v_2)$) and node pair distances ($dist(u_1, v_1)$, $dist(u_1, v_2)$, $dist(u_2, v_1)$, $dist(u_2, v_2)$), which means they can be efficiently computed. Moreover, as they give an upper bound for the maximum point distances between two edges, they can be used to estimate the coverage number upper bound on an edge. Specifically, given an edge e, we identify the set

E_{can} of candidate edges, such that for each edge $e' \in E^e_{can}$, we $UB(e, e') \leq \gamma$. Apparently, if an edge $\hat{e} \notin E^e_{can}$, it means no matter which location in e is the station placed, it cannot cover the spatial objects in $\hat{e}$, as their distance would be larger than γ. We can simply add up the number of spatial objects in every $e' \in E^e_{can}$ as the coverage number upper bound.

The Searching Algorithm

With the upper bound defined, we can search the graph in a greedy manner. The edges with higher upper bounds are searched first. The search is terminated if the upper bounds of the remaining edges are less than the coverage number of the current optimal point. Algorithm 3 shows the pseudocode for searching the whole graph. It takes as input the road network $G(V, O, E)$ and a radius γ, and outputs the station location that covers the maximum number of spatial objects. It uses a priority queue h to keep the edges and their coverage number upper bound and h is initialized as empty (line 1). Initially, for each edge e, it computes the coverage number upper bound and pushes it into the priority queue (lines 2–5). Specifically, it first identifies the set E^e_{can} of edges such that the distance upper bound $UB(e, e')$ does not exceed γ for every $e' \in E^e_{can}$ (line 3). Then it computes the coverage number upper bound by adding up the number of objects on every edge in E^e_{can} (line 4). With all upper bounds computed and pushed into h, it starts searching greedily (lines 7–14). It uses p_{opt} to record the currently known optimal location and cn_{opt} to record its coverage number. In each iteration, it pops edge e with the maximum upper bound from h (line 8). If the upper bound is less than cn_{opt}, it is unnecessary to examine the remaining edges and the search is terminated (liens 9–10). Otherwise, it constructs the corresponding *MOI* problem on the edge e by generating the segments on e from the objects on E^e_{can} (line 11). Then it invokes Algorithm 2 to solve the *MOI* problem and find the optimal point p and its coverage number cn on the edge e (line 12). The current optimal point p_{opt} is updated accordingly if $cn > cn_{opt}$ (lines 13–14). When the search is finished, it returns p_{opt} and cn_{opt} to the users as the final result.

Remark 2. *In Algorithm 3, in order to compute the coverage number upper bound for each edge, we need to identify the edge set E^e_{can} (line 3). For each edge $e' \in E^e_{can}$, we compute $UB(e, e')$, which involves many node pair distance computations. As an edge e' can only be added into E^e_{can} if $UB(e, e') \leq \gamma$, we do need to compute all pair-wise distances in the road network. In contrast, we only need to perform a length-constrained Dijkstra algorithm for each node, i.e., compute the set of nodes whose distance is less than γ. This is much more efficient.*

Algorithm 3: MCRExact.

Input: A road network $G(V, O, E)$, a radius γ
Output: The location p of a station that covers the maximum number of objects.

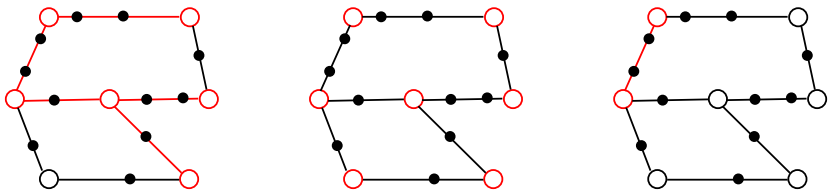
1 $h \leftarrow$ a priority queue;
2 **for** each edge $e = (u, v) \in E$ **do**
3 $\quad E^e_{can} = \{e' | UB(e, e') \leq \gamma\}$;
4 $\quad cn_{max} \leftarrow \sum_{e' \in E^e_{can}} O(e')$;
5 $\quad h.push((cn_{max}, e))$;
6 $cn_{opt} = 0, p_{opt} = null$;
7 **while** h is not empty **do**
8 $\quad cn_{max}, e \leftarrow h.pop()$;
9 $\quad$ **if** $cn_{max} < cn_{opt}$ **then**
10 $\quad\quad$ **break**
11 $\quad S_e \leftarrow$ segments on the edge e generated by the objects in E^e_{can};
12 $\quad I, cn \leftarrow$`MOILinearScan(`S_e`)`, $p \leftarrow$ a random point on I;
13 $\quad$ **if** $cn > cn_{opt}$ **then**
14 $\quad\quad cn_{opt} \leftarrow cn, p_{opt} \leftarrow p$;
15 **return** p_{opt}

Complexity. To compute the coverage upper bound, we need to perform a length-constrained Dijkstra algorithm for every node. Let $O(n')$ be the average complexity for such a length-constrained Dijkstra algorithm. It takes $O(|V| \cdot n')$ to initialize the priority queue h. During the search, we need to compute the segments on the edge and invoke Algorithm 2 to solve the constructed *MOI* problem. Let $O(s')$ be the number of generated segments. It takes $O(s' \log s')$ to finish one iteration. Let k be the number of iterations that we need to search. The total time complexity is $O(|V| \cdot n' + k \cdot s' \log s')$.

2.3. An Approximate Solution

In the previous section, we demonstrated an exact solution to the MCR problem. We notice that the complexity of the exact solution increases when the radius becomes larger, as we visit more nodes in the length-constrained Dijkstra algorithm. In some scenarios, people may focus more on efficiency than accuracy. They may intend to sacrifice a little accuracy for much better efficiency. As a result, in this section, we present two approximate solutions. Though they may not always return the exact result, they can still find satisfactory results and are highly efficient.

Before we present the approximate solutions, we first analyze the search space of the exact algorithm *MCRExact*. Recall that we computed the upper bound for each edge in the network. The edges are searched in a greedy manner until the upper bound of the remaining edges is smaller than the current result. As shown in Figure 7a, the edges that are searched are marked in red. Apparently, it takes most of the runtime to search the edges. There are infinite candidate locations on each edge. Even though we constructed a corresponding *MOI* problem and reduce the search space from infinity to $O(|S|)$, it is still inefficient when there are many spatial objects. Moreover, when the network consists of many edges, the total runtime will be much longer. Thus, to achieve better efficiency, we propose two approaches, namely *NodeApprox* and *EdgeApprox*, which will be elaborated on next.



(a) MCRExact (b) NodeApproximate (c) EdgeApproximate

Figure 7. The high-level idea of the exact approach and the two approximate approaches. The search space of each approach is highlighted in red.

The NodeApprox approach. Instead of searching the edges in a greedy manner, the *NodeApprox* approach adopts a totally different idea. Specifically, it only considers the nodes in the network as the candidate locations, i.e., the station can only be placed on a node in the road network. As shown in Figure 7b, only the nodes in the road network are marked in red, which means we will only examine every node and find the node that covers the maximum number of spatial objects. To examine a node, we can invoke a length-constrained Dijkstra algorithm to obtain the number of covered spatial objects.

The complexity of this approach is $O(|V| \cdot n')$, where n' is the average complexity for the length-constrained Dijkstra search, and $|V|$ is the number of nodes in the road network.

The EdgeApprox Approach. The *NodeApprox* approach only considers the nodes in the road network as candidate station locations. It totally disregards the candidate locations on the edges, which makes it difficult to find satisfactory results in some scenarios. To tackle this issue, we design *EdgeApprox* with the following ideas: We first apply a length-constrained Dijkstra search for each node in the road network. Let $cn(v)$ be the coverage number of node v, i.e., the number of spatial objects that v can reach within the radius. For each edge (u, v), we compute its score as $Score((u, v)) = cn(u) + cn(v)$. Intuitively, if

the two nodes have large scores, it is likely that the edge connecting the two nodes will contain a satisfactory location for the station. We pick the edge with the maximum score and construct a *MOI* problem on this edge. We invoke Algorithm 2 to solve the constructed *MOI* problem and return the best point on this edge to the user as the approximate result.

The complexity of this approach is $O(|V| \cdot n' + s \log s)$, where $|V|$ is the number of nodes, n' is the average complexity for the length-constrained Dijkstra search, and s is the number of segments in the constructed *MOI* problem.

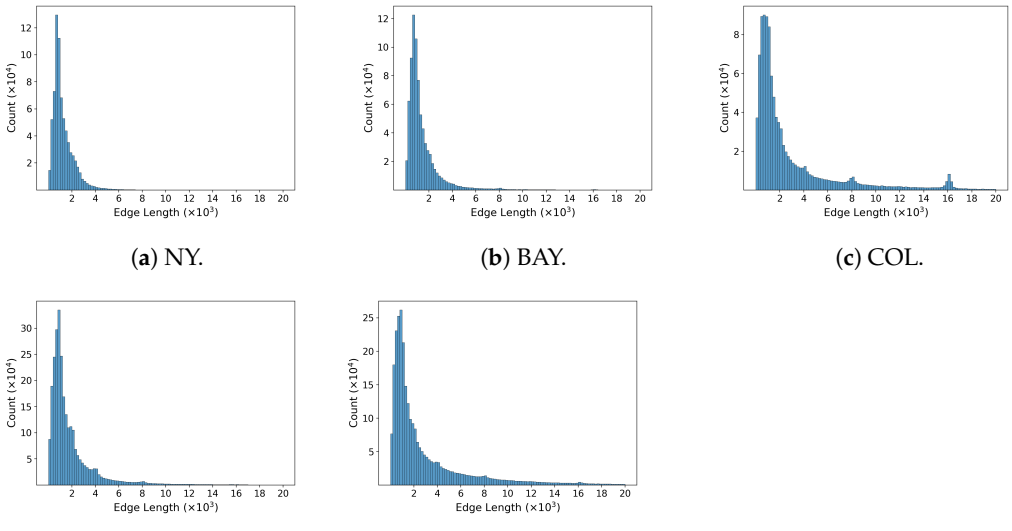
3. Results and Discussion

3.1. Experimental Setup

Datasets. In this paper, we use five real-world public datasets, namely NY, BAY, COL, FLA, and NW (http://www.diag.uniroma1.it//challenge9/download.shtml (accessed on 31 January 2023)). These datasets are the road networks in New York City, San Francisco Bay Area, Colorado, Florida, and Northwest USA, respectively. In the road network, each edge represents a road segment and its weight is the physical distance of the road. For each dataset, we randomly generate spatial objects and place them on the road network. Specifically, the number of spatial objects on an edge is proportional to $p((u,v)) \propto deg(v) \cdot deg(u) \cdot dist(u,v)$, where $deg(v)/deg(u)$ is the degree of node u/v, and $dist(u,v)$ is the length of the edge. Informally, if an edge is connected to nodes with a higher degree, or if the edge is longer, it is likely to hold more spatial objects. Each edge holds about 15.2 spatial objects on average. The details of the five datasets are reported in Table 1. We illustrate the distribution of edge lengths in each dataset in Figure 8. As can be observed from the figure, the lengths of most of the edges are smaller than 6000.

Table 1. Details of the datasets. # nodes, # edges, and # objects represent the numbers of nodes, edges, and objects, respectively.

Dataset	# Nodes	# Edges	Avg. Edge Length	Median of Edge Lengths	# Objects
NY	264,346	733,846	1293.30	951	11,191,261
BAY	321,270	800,172	1630.17	992	12,202,944
COL	435,666	1,057,066	3499.81	1400	16,118,109
FLA	1,070,376	2,712,789	2043.46	1148	41,366,187
NW	1,207,945	2,840,208	3335.88	1449	43,309,864



(**a**) NY. (**b**) BAY. (**c**) COL.

(**d**) FLA. (**e**) NW.

Figure 8. Distribution of edge length in each dataset.

Evaluated Algorithms. We evaluate the following algorithms. (a) *MCRExact*, the exact algorithm for the MCR problem; (b) *NodeApprox*, the method that only considers nodes in the road network as the candidate locations for the station; (c) *EdgeApprox*, the method that only search the most promising edge.

Query Radius. Recall that the proposed approach involves many length-constrained Dijkstra searches. Apparently, the edge length in the road network influences the efficiency of the algorithm. Thus, it is not a good idea to use the same query radius on different datasets. To consider the effect of edge length, we set $\tau = 0.5 \cdot$ avg. edge length as the unit query radius, where *avg. edge length* is the average edge length of the road network. We set the query radius as $\gamma = k\tau$. We can change the query radius by using different values for k.

Performance Measure. We consider the following performance measures: (1) the runtime of each algorithm and (2) the coverage number, i.e., the number of spatial objects that can be covered if the station is placed at the location returned by the algorithm.

All algorithms were implemented in Python 3.9.7. All experiments were run on a Windows desktop with AMD Ryzen 7 5800H CPU and 16 GB memory.

3.2. Experimental Results

The efficiency of the proposed approaches. In this set of experiments, we evaluated the efficiency of the proposed approaches on different datasets by varying the query radius. The runtimes of the evaluated algorithms are reported in Table 2. From the table, we can see that the proposed approximate solutions are much more efficient than the exact solution, especially when the radius is small. This is because when the radius is large, the coverage of a station is more likely to cover complete edges, which makes the upper bound more tight. With a tighter upper bound, more edges can be pruned. In some cases, the runtimes of the approximate solutions are about $\frac{1}{8}$ of that of the exact solution. This illustrates that the proposed approximate approaches are much more efficient than the exact algorithm.

Table 2. Runtime of the proposed approach vs. query radius (seconds).

	Method	**Radius**			
NY		500	1000	1500	2000
	MCRExact	40.37	46.12	18.48	19.67
	NodeApprox	5.05	8.85	12.54	18.18
	EdgeApprox	5.35	9.21	13.29	18.34
	Method	**Radius**			
BAY		500	1000	1500	2000
	MCRExact	48.46	48.43	48.23	39.90
	NodeApprox	5.47	8.76	12.02	16.28
	EdgeApprox	5.91	9.34	12.57	16.52
	Method	**Radius**			
COL		1500	3000	4500	6000
	MCRExact	80.63	45.34	38.94	56.29
	NodeApprox	11.77	21.82	34.49	49.63
	EdgeApprox	12.27	22.33	35.38	50.70
	Method	**Radius**			
FLA		1000	2000	3000	4000
	MCRExact	126.84	70.95	104.18	139.55
	NodeApprox	27.18	48.15	74.20	102.48
	EdgeApprox	28.86	49.44	76.55	104.92
	Method	**Radius**			
NW		1500	3000	4500	6000
	MCRExact	116.94	64.97	86.47	125.33
	NodeApprox	32.05	57.96	91.57	129.75
	EdgeApprox	34.08	59.99	94.09	132.28

Quality of Results of the Approximate Solutions. We will now investigate the quality of results of the approximate solutions. We vary the coverage radius and report the number of covered spatial objects when the station is placed at the location returned by the solution in Table 3. We observe that the approximate solutions return a comparative result as the exact solution. For instance, in most datasets, the results of the two approximate algorithms are about 98% of the results of the exact algorithm. In addition, we notice that when the radius is larger, the gap between the quality of results of the approximate solution and the exact solution is smaller.

Table 3. Result Quality of the approximate approach vs. query radius.

	Method	**Radius**			
NY		500	1000	1500	2000
	MCRExact	276	651	1172	1815
	NodeApprox	275	648	1142	1765
	EdgeApprox	245	650	1152	1815
	Method	**Radius**			
BAY		500	1000	1500	2000
	MCRExact	224	443	719	1133
	NodeApprox	223	442	718	1110
	EdgeApprox	195	442	718	1126
	Method	**Radius**			
COL		1500	3000	4500	6000
	MCRExact	331	1085	2224	3375
	NodeApprox	330	1082	2223	3349
	EdgeApprox	330	1080	2224	3375
	Method	**Radius**			
FLA		1000	2000	3000	4000
	MCRExact	453	1162	1716	2451
	NodeApprox	451	1160	1713	2447
	EdgeApprox	453	1162	1707	2448
	Method	**Radius**			
NW		1500	3000	4500	6000
	MCRExact	486	1453	2981	4802
	NodeApprox	480	1447	2957	4777
	EdgeApprox	480	1450	2980	4802

Effectiveness of upper bound. Recall that in the exact solution, we design an upper bound for each edge so that we can search the edges in the graph in a greedy manner. In this set of experiments, we investigate the effectiveness of the upper bound.

Firstly, Table 4 reports the number of edges that are searched in the road network and its ratio to the total number of edges. A smaller value means that many edges are pruned by their upper bounds. We observe that only a small ratio of edges are searched. In most cases, less than 1% of edges are searched. This demonstrates the effectiveness of our proposed pruning technique. In addition, as can be observed from the table, the ratio of the edges that are searched decreases as the radius increases. This is because more complete edges are fully covered when the radius is large. Recall that in the upper bound computation, we consider all spatial objects on both fully-covered edges and partially covered edges. When there are more fully-covered edges, the upper bound is tighter, which means we can prune more edges.

Secondly, we compare the runtime of the exact solution when the upper bound-based pruning technique is removed, i.e., we do not terminate the algorithm early and keep searching until all edges have been examined. Figure 9 reports the runtime of the exact solution with and without the upper bound-based pruning. As can be observed from

the figure, when there is no pruning, the runtime increases as the radius increases. The gap becomes more obvious when the radius is large. When the radius is 2000, the exact algorithm with pruning is about an order of magnitude faster than the counterpart without pruning. This demonstrates the effectiveness of the upper bound.

Table 4. Effectiveness of the pruning technique based on the upper bound. The table reports the number of edges that are searched, and the ratio to the total number of edges.

NY	Radius	500	1000	1500	2000
	#examined edge	82,614	37,777	4739	2243
	Ratio	11.26%	5.15%	0.65%	0.31%
BAY	Radius	500	1000	1500	2000
	#examined edge	102,053	53,284	32,089	14,749
	Ratio	12.75%	6.66%	4.01%	1.84%
COL	Radius	1500	3000	4500	6000
	#examined edge	109,971	14,911	1751	1736
	Ratio	10.40%	1.41%	0.17%	0.16%
FLA	Radius	1000	2000	3000	4000
	#examined edge	133,825	18,203	16,395	15,455
	Ratio	4.93%	0.76%	0.60%	0.57%
NW	Radius	1500	3000	4500	6000
	#examined edge	115,775	5886	477	311
	Ratio	4.08%	2.07%	0.02%	0.01%

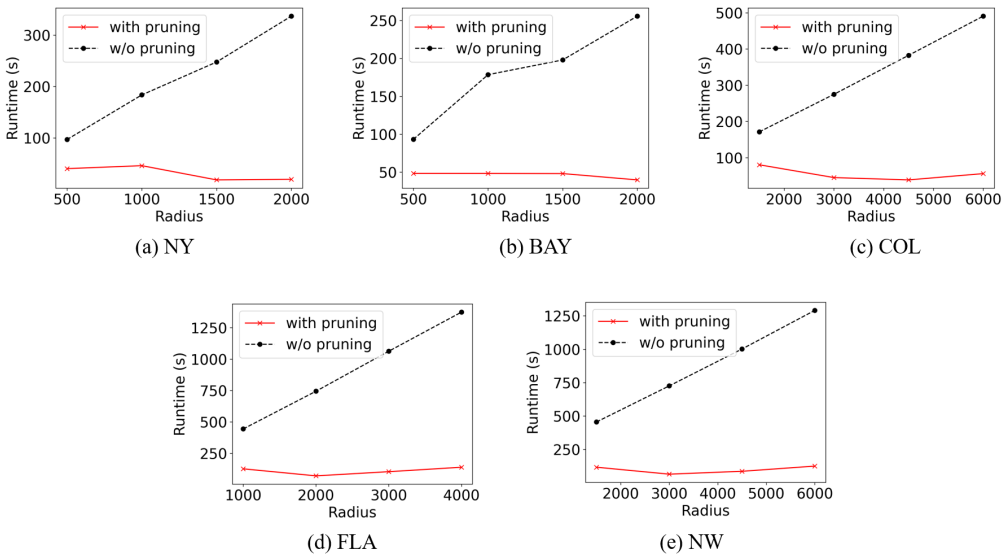
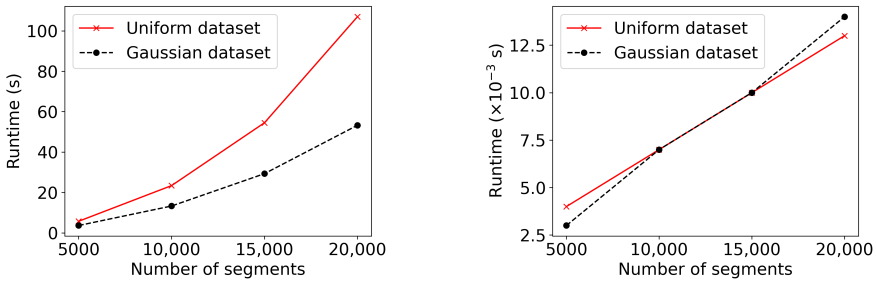


Figure 9. The effect of upper-bound-based pruning on runtime.

The efficiency of *MOIEnumerate* and *MOILinearScan*. Recall that in order to solve the constructed *MOI* problem, we designed two algorithms, namely *MOIEnumerate* and *MOILinearScan*. In this set of experiments, we investigated the efficiency of the two algorithms. Specifically, we randomly generated segments on an edge to obtain two types of instances of the *MOI* problem: (1) In the first type, the segments were generated by following the uniform distribution, i.e., the endpoints of the segments were randomly drawn from uniform distribution independently. Thus, the segment length was not fixed. (2) In the second type, the length of the segments was fixed. The center of each segment was

randomly drawn from a normal distribution. Figure 10 reports the runtimes of *MOIEnumerate* and *MOILinearScan* w.r.t. the number of segments. From the figure, we observe that (1) the runtime of MOIEnumerate increases rapidly as the number of segments increases. This is consistent with its complexity, which is $O(|S|^2)$, where $|S|$ is the number of segments in the *MOI* problem. (2) MOILinearScan scales well and is almost linear w.r.t. the number of segments. The results show that the proposed Algorithm 2 is highly efficient.



(**a**) Runtime of MOIEnumerate. (**b**) Runtime of MOILinearScan.

Figure 10. Runtime of MOIEnumerate and MOILinearScan vs. the number of segments.

Scalability. Finally, in this set of experiments, we evaluated the scalability of the three proposed approaches. We used the largest road network NW and increased the number of spatial objects in the road network from 2,200,849 to 107,215,000. We used a fixed query radius of 1500 and report the runtime of the three approaches in Figure 11. As can be seen from the figure, all three approaches scaled well with respect to the number of spatial objects. This demonstrates the superiority of our proposed approaches.

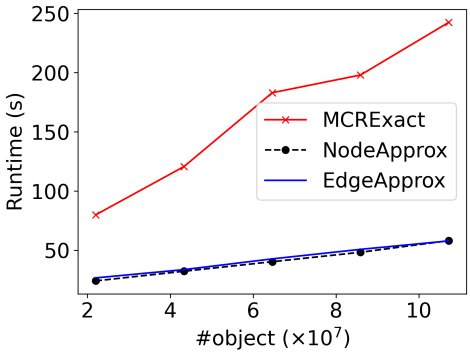


Figure 11. Scalability.

4. Conclusions

In this paper, we propose a novel problem named *max coverage region* (MCR). The problem aims to select the optimal location for a station in the road network, such that the station can cover as many spatial objects as possible. There are an infinite set of candidate locations in the road network. To tackle this challenge, we propose the *most overlapped interval* (MOI) problem and show that we can solve the MCR problem by solving the *MOI* problem. With this reduction, we propose a linear scan-based algorithm to efficiently solve *MOI* in $O(|S| \log |S|)$ time. We further propose an edge-level upper bound estimation method to prune unnecessary edges to improve the efficiency. In addition to the exact solution, we propose approximate solutions to sacrifice a little accuracy for much better efficiency. We conducted extensive experiments on five real-world road networks and demonstrate the effectiveness and efficiency of the proposed solutions.

Author Contributions: Conceptualization, L.F. and Y.Z. (Yudong Zhang); methodology, L.F., G.Y.Y. and Y.Z. (Yudong Zhang); software, L.F.; validation, L.F., Z.K., Y.Z. (Yuzhang Zhou) and Y.Z. (Yudong Zhang); formal analysis, L.F., Z.K., Y.Z. (Yuzhang Zhou) and Y.Z. (Yudong Zhang); investigation, L.F.; resources, L.F., Z.K., Y.Z. (Yuzhang Zhou) and Y.Z. (Yudong Zhang); data curation, L.F., Z.K., Y.Z. (Yuzhang Zhou) and Y.Z. (Yudong Zhang); writing—original draft preparation, L.F.; writing—review and editing, L.F., G.Y.Y. and Y.Z. (Yudong Zhang); visualization, L.F., Z.K., Y.Z. (Yuzhang Zhou) and Y.Z. (Yudong Zhang); supervision, L.F.; project administration, L.F.; funding acquisition, Y.Z. (Yudong Zhang). All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (grant no. 61906039), Youth Scholar Program of SEU, and the Fundamental Research Funds for the Central Universities (grant no. 2242022k30007), Medical Research Council Confidence in Concept Award, UK (MC_PC_17171); Royal Society International Exchanges Cost Share Award, UK (RP202G0230); British Heart Foundation Accelerator Award, UK (AA/18/3/34220); Hope Foundation for Cancer Research, UK (RM60G0680); Global Challenges Research Fund (GCRF), UK (P202PF11); Sino-UK Industrial Fund, UK (RP202G0289); LIAS Pioneering Partnerships award, UK (P202ED10); Data Science Enhancement Fund, UK (P202RE237); Fight for Sight, UK (24NN201); Sino-UK Education Fund, UK (OP202006); Biotechnology and Biological Sciences Research Council (BBSRC), UK (RM32G0178B8).

Data Availability Statement: The five real-world road network datasets (NY, BAY, COL, FLA, and NW) comes from https://www.diag.uniroma1.it/challenge9/download.shtml (accessed on 12 May 2022).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nandy, S.C.; Bhattacharya, B.B. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Comput. Math. Appl.* **1995**, *29*, 45–61. [CrossRef]
2. Imai, H.; Asano, T. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms* **1983**, *4*, 310–323. [CrossRef]
3. Choi, D.W.; Chung, C.W.; Tao, Y. A scalable algorithm for maximizing range sum in spatial databases. *arXiv* **2012**, arXiv:1208.0073.
4. Tao, Y.; Hu, X.; Choi, D.W.; Chung, C.W. Approximate MaxRS in spatial databases. In Proceedings of the 39th International Conference on Very Large Data Bases 2013, (VLDB 2013), Riva del Garda, Italy, 26–30 August 2013; Volume 6, pp. 1546–1557.
5. Feng, K.; Cong, G.; Bhowmick, S.S.; Peng, W.C.; Miao, C. Towards best region search for data exploration. In Proceedings of the 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 1055–1070.
6. Mostafiz, M.I.; Mahmud, S.F.; Hussain, M.M.U.; Ali, M.E.; Trajcevski, G. Class-based conditional MaxRs query in spatial data streams. In Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, 27–29 June 2017; pp. 1–12.
7. Feng, K.; Cong, G.; Jensen, C.S.; Guo, T. Finding attribute-aware similar region for data analysis. *Proc. VLDB Endow.* **2019**, *12*, 1414–1426. [CrossRef]
8. Kalinin, A.; Cetintemel, U.; Zdonik, S. Interactive data exploration using semantic windows. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, UT, USA, 22–27 June 2014; pp. 505–516.
9. Kalinin, A.; Cetintemel, U.; Zdonik, S. Searchlight: Enabling integrated search and exploration over large multidimensional data. *Proc. VLDB Endow.* **2015**, *8*, 1094–1105. [CrossRef]
10. Cao, X.; Cong, G.; Jensen, C.S.; Yiu, M.L. Retrieving regions of interest for user exploration. *Proc. VLDB Endow.* **2014**, *7*, 733–744. [CrossRef]
11. Zhang, D.; Du, Y.; Xia, T.; Tao, Y. Progressive computation of the min-dist optimal-location query. In Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Republic of Korea, 12–15 September 2006; pp. 643–654.
12. Xiao, X.; Yao, B.; Li, F. Optimal location queries in road network databases. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, Hannover, Germany, 11–16 April 2011; pp. 804–815.
13. Chen, Z.; Liu, Y.; Wong, R.C.W.; Xiong, J.; Mai, G.; Long, C. Efficient algorithms for optimal location queries in road networks. In Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, Snowbird, UT, USA, 22–27 June 2014; pp. 123–134.
14. Du, Y.; Zhang, D.; Xia, T. The optimal-location query. In Proceedings of the International Symposium on Spatial and Temporal Databases, Angra dos Reis, Brazil, 22–24 August 2005; pp. 163–180.
15. Cabello, S.; Díaz-Báñez, J.M.; Langerman, S.; Seara, C.; Ventura, I. *Reverse Facility Location Problems*; Citeseer: Princeton, NJ, USA, 2006.
16. Wong, R.C.W.; Özsu, M.T.; Yu, P.S.; Fu, A.W.C.; Liu, L. Efficient method for maximizing bichromatic reverse nearest neighbor. *Proc. VLDB Endow.* **2009**, *2*, 1126–1137. [CrossRef]

17. Choudhury, F.M.; Culpepper, J.S.; Sellis, T.; Cao, X. Maximizing bichromatic reverse spatial and textual k nearest neighbor queries. *Proc. VLDB Endow.* **2016**, *9*, 456–467. [CrossRef]
18. Zhou, Z.; Wu, W.; Li, X.; Lee, M.L.; Hsu, W. Maxfirst for maxbrknn. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, Hannover, Germany, 11–16 April 2011; pp. 828–839.
19. 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, Glasgow, UK, 24–28 October 2011; pp. 1475–1484.
20. Huang, J.; Wen, Z.; Qi, J.; 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, Glasgow, UK, 24–28 October 2011; pp. 2377–2380.
21. Xia, T.; Zhang, D.; Kanoulas, E.; Du, Y. On computing top-t most influential spatial sites. In Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, 30 August–2 September 2005; pp. 946–957.
22. Zhan, L.; Zhang, Y.; Zhang, W.; Lin, X. Finding top k most influential spatial facilities over uncertain objects. In Proceedings of the the 21st ACM International Conference on Information and Knowledge Management, Maui, HI, USA, 29 October–2 November 2012; pp. 922–931.