Geoinformatica
https://doi.org/10.1007/s10707-019-00343-4

Top-k trajectories with the best view

Nafis Irtiza Tripto[1] · Mahjabin Nahar[1] · Mohammed Eunus Ali[1] · Farhana Murtaza Choudhury[2] · J. Shane Culpepper[2] · Timos Sellis[3]

Received: 18 April 2018 / Revised: 7 December 2018 / Accepted: 13 February 2019 /
Published online: 26 February 2019

Abstract

The widespread availability of GPS and the growing popularity of location based social networking applications such as Flickr, Yelp, etc., enable more and more users to share their route activities or trajectories. At the same time, the recent advancement in large-scale 3D modeling has inspired applications that combine visibility and spatial queries, which in turn can be integrated with user trajectories to provide answers for many real-life user queries, such as *"How can I choose the route which provides the best view of a historic site?"*. In this work, we propose and investigate the k Aggregate Maximum Visibility Trajectory (kAMVT) query and its variants. Given sets of targets, obstacles, and trajectories, the kAMVT query finds top-k trajectories that provide the best view of the targets. We extend the kAMVT query to incorporate different weights (or preferences) with trajectories and targets. To provide an efficient solution to our problem, we employ obstacle and trajectory pruning mechanisms. We also employ an effective target ordering technique, which can further improve query efficiency. Furthermore, we extend the proposed queries to introduce preferences on trajectories in situations where smaller trajectories are preferred due to time constraints, or trajectories closer to the query user are preferred. To verify the efficiency and effectiveness of our solutions, we conduct an extensive experimental study using large synthetic and real datasets.

Keywords Spatial databases · Query processing · Obstacles · Trajectories · Visibility

✉ Mohammed Eunus Ali
eunus@cse.buet.ac.bd

Farhana Murtaza Choudhury
farhanamc@rmit.edu.au

Timos Sellis
tsellis@swin.edu.au

[1] Bangladesh University of Engineering and Technology, Dhaka, Bangladesh

[2] RMIT University, Melbourne, Australia

[3] Swinburne University of Technology, Melbourne, Australia

1 Introduction

With the widespread use of GPS-equipped mobile devices and popular map services, an unprecedented amount of trajectory data is becoming available. For example, in Bikely[1] users can share their cycling routes from GPS devices; in GPS-wayPoints[2] a user can add waypoints (points on a route where a course has changed) in a route and share with friends; users can share their travel routes and experience using GPS trajectories in Microsoft GeoLife.[3] Most of the popular social network sites, e.g., Flickr, Yelp, Twitter, Foursquare, and Facebook also support sharing user trajectories.

People explore and describe locations and travel experiences with a strong emphasis on visual senses. While the increasing availability of large-scale 3D models from mapping services, such as Google Maps, Google Earth, OpenStreetMap, and indoor 3D mapping services such as Google Tango inspire the applications involving visibility in spatial databases, applications involving visibility in trajectories has received little attention in the literature.

Measuring visibility from travel routes and ranking the routes have many applications in tourism, trip planning, advertisement, and businesses, for example:

- **Trip planning:** Consider a scenario where a tourist plans to visit historic sites in Rome, but is unsure which routes would provide the best views of these sites. In real life, visual obstacles such as buildings in the city may completely or partially block the view of the historic sites of interest. Routes followed by the previous tourists, which are "historical" trajectories in the target area can provide some guidance on which routes are the best.
- **Virtual reality (VR) based trip planning:** Consider a scenario that integrates a VR device with Google Earth to offer a tourist a visual guide using an augmented heat map. The heat map essentially assigns different colors to different road segments (or parts of trajectories) based on the visibility of the selected historic sites (targets) that the tourist wants to visit. Then, the tourist can select a path that would be most likely to provide the best visual experience from a potential set of trajectories based on this heat map.
- **Indoor route planning:** The Google Tango device can now capture 3D models of indoor spaces. Consider a scenario of a large Museum, where the layouts of the entire indoor space including the artifacts that are on display have been captured. The Museum authority also regularly captures the paths of each individual visiting the Museum as a trajectory. Walking around each artifact to find the best possible view is not always feasible for a tourist in a limited time constraint. In this scenario, a guided tour path can be presented to a new visitor that includes the desired artifacts with an improved visual experience from historical paths of previous visitors.

Example 1 Figure 1 illustrates an example scenario, where the user u wants to visit two target sites t_1 and t_2 and the database contains historical trajectory data l_1, l_2, and l_3 of the previous users. Each trajectory consists of a sequence of geo-spatial data points. Here, $o_1, o_2, \ldots, o_7$ are different obstacles located around the site. For each target, we consider a visible range around it (shown with a dotted circle) assuming that the target is not visible from any trajectories outside of this visible range.

[1] http://www.bikely.com

[2] http://gpswaypoints.net

[3] https://research.microsoft.com/en-us/projects/geolife/

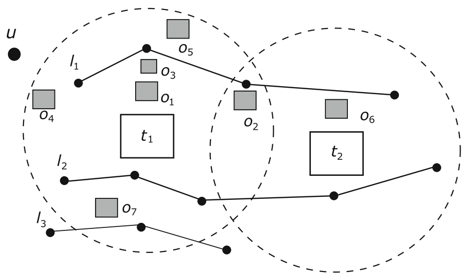
Fig. 1 An example of the *k*AMVT query

Assume that u wants the best visibility trajectory for the target t_1. For t_1, certainly l_2 provides the maximum visibility, since the views from both l_1 and l_3 are obstructed. Therefore, l_2 is the best visibility scored trajectory with respect to t_1.

Now consider the case where u wants the top-2 trajectories with the best visibility of both targets t_1 and t_2. It is clear that both t_1 and t_2 are visible from l_1 and l_2, but only t_1 is visible from l_3 since it is outside of the visible range of t_2. Suppose, the user wants to find the top two trajectories based on the combined visibility of both target objects. Here, l_1 and l_2 are the candidates for the best combined visibility, since they provide some visibility for both targets. Therefore, the result will be (l_2, l_1).

In the application scenarios and the example, the underlying problem is to find the k best trajectories that provide the aggregated visibility of a set of query target objects in an obstructed space. In real applications, the trajectories can be associated with weights based on different criteria. For example, (i) if a trajectory with fewer segments and shorter length is more preferable, then a higher weight can be assigned to such trajectories (l_3 is more preferable considering t_1); (ii) if the trajectories closer to a specific location (the user's current location for example) is preferable, the weight of each trajectory can be assigned according to the distance from that specific location (l_1 is more preferable since it nearer to the location of user u). Moreover, the preference for each target object can be different for a user.

In this paper, we address the problem of finding a limited number of weighted trajectories with the maximum aggregated visibility of a set of query target objects in an obstructed space. We denote this problem as the k Aggregate Maximum Visibility Trajectory (kAMVT) query. Formally, given a set of trajectories L where each trajectory is associated with a weight, a set of obstacles O, a set of query target objects T, an aggregation function f, a preference vector P over T, and a positive integer k, kAMVT returns k trajectories from L that provide the maximum aggregated visibility of T according to function f and the preference vector P.

The visibility problem has been extensively studied in the fields of Computational Geometry and Computer Graphics by constructing visibility polygons and visibility graphs. However, these algorithms rely heavily on accessing all obstacles and/or preprocessing of data, which is unsuitable in spatial applications due to the dynamic nature of data and the time constraints [1–3, 18, 38, 46]. Moreover, visibility is also considered as a Boolean notion (visible/ not visible) in several studies [4, 37, 42]. Masud et al. [24] introduce a query called the Maximum Visibility (MV) query which finds the location that maximizes the visibility of the target object. Since an MV query [24] considers a single target only, and computes visibility from point locations, their methods cannot be directly extended to solve the maximum visibility problem for trajectories. Also the problem becomes more challenging for multiple targets. Rabban et al. [30] introduce a Visibility Color Map (VCM), where the space is partitioned, and a heat-map of the space is generated by assigning a color to each partition according to the visibility of a pre-defined target. However, this approach cannot be directly extended to our problem which ranks trajectories according to visibility w.r.t. multiple target objects.

Although several studies have addressed the problem of ranking trajectories and returning the top ones based on different spatial similarity metrics, the visibility of an object introduces additional challenges. The visibility of a target object with respect to a user trajectory depends not only on location, but also on other objects (visual obstacles) between them. In addition, since visibility is commonly defined with respect to a point location, one challenge is to define and compute visibility with respect to a sequence of line segments (the trajectories).

In this paper, we introduce and address the *k* Aggregate Maximum Visibility Trajectory query. As in real applications this query can have many variations (weighted vs. non-weighted trajectories, single vs. multiple target objects), and we present a general framework to address the *k*AMVT and its variants. We propose several techniques to efficiently answer the *k*AMVT query. Specifically, the technical contributions of our solution include:

- **Trajectory partitioning:** Due to the presence of obstacles, different portions of a target can be completely or partially obstructed from different points over a trajectory. To compute how well a target object is visible from a trajectory, the idea is to partition a trajectory in a visually meaningful way such that for each partitioning, the visible part of a target from each of the points of the partition can be considered as the same.
- **Obstacle Pruning:** The intuition behind the idea of obstacle pruning is that the obstacles that are shadowed by other obstacles will not affect the visibility of the target. In this way, many obstacles can be pruned.
- **Trajectory Pruning:** If a trajectory previously performed poorly compared to other trajectories, it will not be included in the final list, and can thus be pruned. Besides, if a portion of a trajectory is blocked by an obstacle, that portion can be eliminated from consideration. We achieve this by maintaining an upper and lower bound on the visibility of a trajectory.
- **Efficient Target Selection:** Instead of considering the targets randomly, we can order them according to the number of trajectories that are inside their visibility range. In this way, we can eliminate many trajectories from consideration, and in some cases, can eliminate a target altogether.

In summary, the contributions of our paper are as follows:

- We formulate the *k*AMVT query for spatial databases, and propose natural extensions of the problem.
- We define the notion of visibility of a target with respect to a trajectory.
- We introduce efficient approaches to prune a significant number of trajectories and obstacles in the spatial databases at query time.
- Finally, we conduct an experimental study on real and synthetic spatial data to compare our proposed solutions.

The rest of paper is organized as follows. Section 2 reviews the related work, Sections 3 and 4 formally define the problem and objective function respectively. In Section 5, we describe our proposed solution. Section 6 presents the experimental results. Finally in Section 7 we provide conclusions and future directions.

2 Related work

The related body of work mostly includes studies in visibility computation in computer graphics and computational geometry, advanced spatial query processing, visibility based spatial query processing, and query processing with trajectories.

2.1 Visibility in computational geometry and computer graphics

Computational geometry and computer graphics literature mostly focus on the binary notion of visibility, where a point or an object is either visible or invisible from another point.

In the field of computational geometry, visibility computations are performed using visibility graphs and visibility polygons [1–3, 18, 38, 46]. A visibility graph is defined in [3] by a set P of n points inside a polygon Q where two points $p, q \in P$ are joined by an edge if the segment $pq \subset Q$. Ben et al. [3] used a visibility graph to propose a near optimal algorithm for simple polygons. They also introduce the concept of robust visibility for polygons with holes, and proposed a near optimal algorithm for that case.

A visibility polygon $V(q)$ of a point q in a polygon P is defined as the set of points in P that are visible from q [46]. Two points p, q inside a polygon are visible to each other if their connecting segment pq remains completely inside the polygon. Zarei and Ghodsi [46] computed the visibility polygon of a query point inside a non-simple polygon by converting it into a simple polygon. Asano et al. [1, 2] later reduced the space requirement and preprocessing time. Although these algorithms solve the problem of constructing a visibility polygon, they are based on preprocessing or accessing all obstacles, which make them inappropriate for many spatial database applications as updates will invalidate the preprocessing steps and accessing all objects is often too expensive.

Visibility computations are also a well studied topic in the area of computer graphics. Visibility maps are often used [4, 37, 42], which are graphs describing a view of the scene including the topology. Visibility maps can be constructed from a fixed viewpoint [4, 37], or from a moving viewpoint [42]. However, in these approaches, visibility is defined from a point source and also the binary notion of visibility is only considered. Kim et al. [20] investigated the problem of visibility of/from an extended region, and determined the subset of the space which is completely visible from a region. However, this method ignores the case of partial visibility of a space.

Existing methods in computer graphics utilize various indexing structures such as an LoD-R-tree or HDoV-tree to support visibility queries in visualization systems [10, 35]. They used various acceleration techniques to render complex models at interactive frame rates. However their main focus was rendering a scene, while our work focuses on calculating the visibility of a set of targets from multiple trajectories, and ranking them based on visibility.

2.2 Spatial query processing in the obstructed space

Spatial query processing such as finding the k nearest neighbors of a query point has been extensively studied in various domains in Euclidean space [19, 26, 36, 39, 40, 45], road network space [21–23, 25, 29, 31], and obstructed space [14, 27, 28, 41, 44].

The concept of k nearest neighbors (kNN) is an extremely versatile concept which is applicable in many spatial problems. The kNN query finds the k nearest points with respect to a query point based on a given similarity measure. In the past, numerous algorithms [36, 39, 40] have been proposed to solve the kNN problem in Euclidean space. Most of these algorithms assumed that the data objects are static, and used tree-based (e.g., R-Tree) structures (or their extensions) to enable efficient query processing. This problem is also a well studied topic in road networks [21, 29], where the distance between a query point and an object is computed in terms of road network distance.

However, in an obstructed space, the distance between two objects can be affected by obstacles. Many variants of kNN queries have been proposed in obstructed space, which include the Obstructed NN (ONN) query [44, 49], the Visible NN (VNN) query [27, 28], the Continuous NN (CNN) query [41], the Continuous Obstructed NN (CONN) query [11], and the Continuous Visible NN (CVNN) query [13, 14].

Given a set of data points P and a query point q in a multidimensional obstructed space, the ONN query finds k data points with minimum obstructed distances to q, where obstructed distance is measured as the length of the shortest path that connects any two points without crossing any obstacle. Zhang et al. [49] studied spatial queries in obstructed space, and proposed efficient algorithms for the most important query types such as, range search, nearest neighbours e-distance joins and closest pairs. In order to deal with huge number of obstacles, they maintain local visibility graphs only for the obstacles that are around the query point q and may influence visibility. Xia et al. [44] propose a solution for finding the kNNs of a given query point according to the obstructed distance. They provide an algorithm to prune irrelevant query points and obstacles by starting with a *local workspace* and introducing obstacles relevant to the query point incrementally.

Nutanong et al. [27, 28] employed the Visible Nearest Neighbors (VNN) search to compute the NN visible to a query point. The basic idea is based on the fact that a farther object cannot affect the visibility of a nearer object and thus, this method performs the NN search and checks its visibility condition incrementally. Later they also provide a variant of VNN, the Aggregate VkNN (AVkNN) query, which finds the visible k nearest objects to a set of query points based on an aggregate distance function [28]. They first proposed an approach that accesses the database via multiple VkNN queries. In order to improve the performance of this approach, they proposed an alternative solution that issues an aggregate kNN query to retrieve objects from the database and then re-rank the results based on the aggregate visible distance.

Tao et al. [41] introduce the Continuous Nearest Neighbor (CNN) query, which finds the kNN for a moving query point. Later, Gao et al. [11] studied a variant of the CNN query in the presence of obstacles, called the Continuous Obstructed Nearest Neighbor (CONN) query. Given a data set P, an obstacle set O, and a query line segment q in a two-dimensional space, the CONN query retrieves the nearest neighbor of each point on q according to the obstructed distance. Their method handles the CONN retrieval by performing a single query for the entire query segment. They also provide a novel concept of control points and an efficient quadratic-based split point computation algorithm to process only the data points and obstacles relevant to the final result.

Similarly, the problem of Continuous Reverse kNN queries and the Continuous Visible Nearest Neighbor (CVNN) queries in obstructed spatial databases has also been studied in [15] and [13, 14], respectively. The CVNN retrieves the visible nearest neighbors in the presence of obstacles. The CVNN extends the CNN [41], and CONN query [11] by taking visibility into consideration.

All the aforementioned NN queries considers the distance or visible distance from a query point. However, our problem requires a visibility computation rather than a distance calculation. Therefore, the approaches used in various NN queries are not applicable in this context.

2.2.1 Visibility based spatial query processing

Visibility based query processing has been a recent focus in the spatial database community due to the availability of large-scale 3D data. Existing studies in computational geometry and spatial databases [28] regard visibility as a binary notion and assume that a point is either visible or not from another point. However, all of the motivating applications mentioned in the introduction require visibility quantification as a continuous notion. This quantification seems important, because a target can be visible from a number of trajectories, but not all of the trajectories have the same coverage of the target.

Masud et al. [24] first introduced the kMVQ query, which finds k locations from a set of query locations that maximize the visibility of a target object T in the presence of obstacles. To address the problem of quantifying the visibility of a target object from the surrounding area, Choudhury et al. [8] proposed a scalable technique to partition the space in a visually meaningful way, and generate a heat-map of the space, called a VCM, that assigns a color to each partition according to the visibility. Later, Rabban et al. [30] improved this technique to incorporate the partial visibility of the target while constructing the VCM. Since both of these approaches deal with the visibility of a target object from the query viewpoint, and our problem includes computing the visibility of a target from multi-point trajectory points, we have adopted this definition of visibility in our work.

Haider et al. [17] address the continuous version of the kMVQ query, denoted as kCMV, where the problem is to continuously report the k locations that maximize the visibility of a moving target. The visibility of T from each query location continuously changes, and the ranked results must be be updated. Their approach consists of a pre-processing step which constructs an aggregated visible region (AVR) and a blocking set (BS) of the query locations.

All previous works consider the visibility of extended objects (i.e., targets) w.r.t. the query point, and it is not straightforward to extend the existing concept to compute the visibility of extended objects from trajectories. Thus in this paper, we propose a comprehensive solution to compute the visibility of targets from trajectories in the presence of a large number of obstacles. We compare existing techniques of visibility based techniques and our proposed technique in Table 1 based on six features: considering different notions of visibility (binary/non binary), computing visibility w.r.t. line/trajectory, pruning obstacles, handling multiple targets (query points), handling moving query points, and ranking of trajectories.

2.2.2 Query processing with trajectories

The increasing availability of user trajectory data have provided new opportunities for trajectory based query processing in spatial database in recent period example as trajectory simplification [48], big trajectory data management [9]. In similarity-based retrieval of moving object trajectories, Chen et al. [5] introduced a novel distance function, Edit Distance on Real sequence (EDR) which is robust against various data imperfections and also developed three pruning techniques to improve the retrieval efficiency of EDR. Later, Chen et al. investigated the problem of searching trajectories that closely match a given set of locations [7]. Given a small set of locations with or without an order specified, this query finds the k-Best-Connected Trajectories (k-BCT) from a database such that the k-BCT best connect the designated locations geographically. In order to achieve the efficiency, they provide a simple Incremental k-NN based Algorithm (IKNN) and some enhancement of this approach.

Table 1 Comparison of visibility based techniques

Techniques	Non-binary visibility	Trajectory based visibility	Obstacle pruning	Multiple targets	Moving query	Trajectory ranking
Visibility polygon [1, 2, 46]	×	×	×	×	×	×
Visibility map [42]	×	×	×	×	✓	×
CVNN [13, 14]	×	×	✓	×	✓	×
*k*MVQ [24]	✓	×	✓	×	×	×
*k*CMV [17]	✓	×	✓	×	✓	×
*k*AMVT	✓	✓	✓	✓	×	✓

Several other works have also focused on finding the best historical trajectories with custom objective functions. The Most Popular Route (MPR) by Chen et al. [6], studies the problem of discovering the MPR between two locations by observing the trajectories of previous users. In another work, Shafique et al. [32] propose a novel technique to find the most popular path within an Region of Interest(ROI) from historical trajectory data by rephrasing trajectories into smaller part and eliminating noisy points from trajectories. Gao et al. proposed mutual nearest neighbor (MNN) query [12] that finds k nearest neighbors of a query object from a given set of trajectories in a query time extent. Later they introduced a constrained region (CR) in their problem and solved the constrained k-nearest neighbor (CkNN) query and historical continuous CkNN (HCCkNN) query on spatial region storing historical information about moving object trajectories.

Recently, the trajectory search using personalized metrics has also been popular and studied in several works. Zheng et al. [50] introduce the problem of efficient similarity search on activity trajectory database. Given a sequence of query locations and a set of desired activities for each location, an activity trajectory similarity query (ATSQ) returns k trajectories that cover the query activities and yield the shortest minimum match distance. Shang et al. [33] propose a novel problem called User Oriented Trajectory Search (UOTS) for trip recommendation, which considers both spatial and textual domains. A pair of upper and lower bounds are maintained to restrain the search range in two domains. In another work, Shang et al. propose a novel problem termed as personalized trajectory matching (PTM) [34]. Given a trajectory with user-specified weights for each sample point in the trajectory, the PTM query returns the trajectory in an argument data set with the highest similarity to the query trajectory. A novel two-phase search algorithm is proposed that carefully selects a set of expansion centers from the query trajectory and exploits upper and lower bounds to prune the search space in the spatial and temporal domains.

None of these approaches consider visibility based query processing for trajectories. Thus, to the best of our knowledge, our work is the first work that integrates the concept of visibility with trajectory based query processing.

3 Problem formulation

Let L be a set of trajectories and O be a set of obstacles in a geo-spatial dataspace. Each trajectory $l \in L$ is a sequence of locations $l.d = \{d_1, d_2, \ldots, d_{|l|}\}$, where each location d_i is represented as a tuple of $\langle longitude, latitude \rangle$. Each obstacle $o \in O$ has a geo-spatial position and extent (line, rectangle, polygon) in the n-dimensional space.

Each trajectory $l \in L$ is optionally associated with a weight $l.w$ (denoting the preference of the trajectory), where the value of $l.w$ depends on the application. For example, the length of the trajectory or the distance of the trajectory from a specific location can correspond to the weight of that trajectory.

Definition 1 Given a set L of weighted trajectories, a set O of obstacles, an aggregation function f, a positive integer k, a set T of target objects, and an associated preference vector P, a kAMVT query returns a set L' of k trajectories from L that provide the maximum visibility of T, where the visibility of the target objects are aggregated by function f according to the preference vector P. Specifically, for any trajectory $l \in L'$ and any trajectory $l' \in L \backslash L'$, $V(T, l) \geq V(T, l')$, where $V(T, l)$ denotes the aggregated visibility of T from the trajectory l.

An example aggregation function is the summation of the visibility of each target, where P corresponding to the preference for each target object in T:

$$V(T, l) = l.w \times \sum_{t \in T} p_t \times V(t, l)$$

Here, a higher value of $l.w$ contributes to a higher value of $V(T, l)$. If the weights of the trajectories are not specified, we consider equal preference for each trajectory. The value of each $p_i \in P$ is between [0, 1] such that $\sum_i p_i = 1$.

4 Preliminaries

In this section we present the visibility computation of a target object from a *point location* as commonly done in the existing literature. As each side of a target is a line in 2D space (face in 3D), without loss of generality, we explain the visibility quantification for a line as the geometric shape in this section.

Previous studies [24, 30, 47] have defined and used different metrics to quantify visibility. The metric, called "visual angle" [24] is the angle formed at the eye level of a user by the extremities of an object viewed, which determines the perceived length of that object. Figure 2 describes this visual angle and we adopt this metric as the measure of visibility in this work. Specifically, the visibility measure of a target t from a point location d is computed as:

$$\mathrm{V}(t, d) = \frac{2 \arctan(\mathrm{PL}(t, d))}{180} \tag{1}$$

where the maximum possible visual angle is 180°, which is used to normalize the value of $\mathrm{V}(t, d)$ between [0,1], and $\mathrm{PL}(t, d)$ is the perceived length of t from d.

In an unobstructed space, the perceived length of an object mainly depends on the distance and the viewing angle between the user and the object. If an object is viewed from an oblique angle, the perceived length of that object becomes smaller than the original length. The perceived length of t also decreases with the increase of the distance between t and the point location d.

Let the straight line connecting the midpoint of the target t (which is a line) and the point location d creates an angle $\angle(t, d)$ with t. Let the minimum distance of target t and the point d be $\mathrm{dist}(t, d)$. Then the perceived length of a t from d is measured as:

$$\mathrm{PL}(t, d) = \frac{\angle(t, d)}{90°} \times \frac{len(t)}{\mathrm{dist}(t, d)} \tag{2}$$

Figure 3 shows an example scenario of calculating simplified visibility for a target. Let ab be a trajectory segment with midpoint c and t be a target line of length 10 units. Assume the

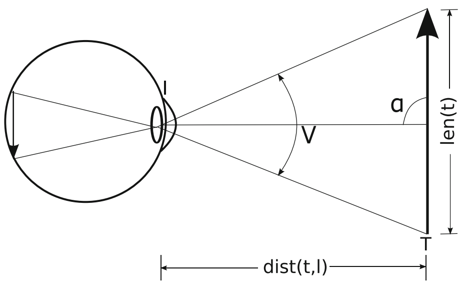

Fig. 2 Computing the visibility from a point location

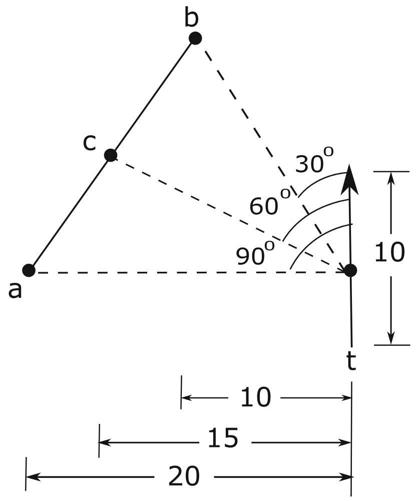

Fig. 3 The visibility of a target from a line segment

minimum distance of t from points a, b, c is 20,15,10 units, and the angle between the points and the midpoint of t is 90°, 60°, 30° respectively. According to Eq. 2, the perceived length of t from point a is measured as $\mathrm{PL}(t, a) = \frac{90^\circ}{90^\circ} \times \frac{10}{20} = 0.5$, and the visibility measure of t is computed as $\mathrm{V}(t, a) = \frac{2\arctan(0.5)}{180} = 0.295$. Similarly, we can compute the visibility of t from b and c, and the simplified visibility of target t from trajectory segment ab is measured as $sim\_vis(ab, t) = \frac{\mathrm{V}(t,a)+\mathrm{V}(t,b)+\mathrm{V}(t,c)}{3} = (0.295 + 0.266 + 0.205)/3 = 0.253$.

5 Our approach

Without loss of generality, we now consider each obstacle $o \in O$ as a rectangular object (rectangular cuboid in 3D) in the data space. The set T of target objects is a query input where each target is also a rectangular object in a data space. The set L of trajectories and the set O of obstacles are stored in separate R-trees [16], where each trajectory $l \in L$ is represented as the Minimum Bounding Rectangle (MBR) of the points of l in the leaf-level entries of the tree. A pointer to the actual trajectory data is associated with the corresponding leaf level node.

Our objective is to find the top-k trajectories from L that provide the best visibility of T based on the effect of the obstacle set O. As there is no prior work that addresses the visibility computation of an object from a continuous sequence of line segments (trajectories), first we present a novel trajectory partitioning approach to compute visibility from trajectories. Then we present a straightforward approach to answer the kAMVT query. In Section 5.3, we present an advanced approach that computes the results of kAMVT by applying several obstacle pruning, trajectory pruning, and target pruning techniques. For ease of presentation, we ignore the weights of the trajectories in the details of the approach presented in this section. Later in Section 5.5 we present how different weights of the trajectories can be incorporated into the solution.

We need to find top-k trajectories from L which provides the best visibility of those targets considering the effect of an obstacle set O. The notation used throughout the paper along with the definitions is listed in Table 2.

Table 2 Notation and definitions

Notation	Definition
T	Set of target objects
t_i	A specific target object
ω	A specific side of the target where $\omega \in \{up, down, left, right\}$
L	Trajectory dataset
L_t	Set of trajectories that are inside the visibility range of target t
l_i	A specific trajectory
d_i	The i·th data point of a trajectory
$S_{t(\omega)}$	The segments that are associated with the specific side ω of the target t
O	obstacle set
O_t	Reduced obstacle set associated with target t
o_i	A specific obstacle
$SS(l_i, t_j)$	Set of segments of trajectory l_i that are inside the visibility range of the target t_j
(s_i, f_i)	Starting and finishing point of trajectory l_i inside the visibility range
P	Prefereance vector associated with target set
k	Total number of trajectories required

5.1 Visibility computation of a target from a trajectory

Here, we discuss how to calculate visibility of a target with respect to a trajectory. Human vision for any given target is limited to a certain range. The visibility decreases as the distance between a viewpoint and a target increases. Therefore for each target, we consider a circular visible range (spherical in 3D) with the assumption that, any obstacles or trajectories that are outside of the visible range have no effect in computing the visibility of the target.

Specifically, visibility pruning has the following logical steps: (i) For a target t and and a trajectory l, we identify the segments of l from which t are visible. If t is partially visible from any part of a segment of l, that segment is further partitioned in a visually meaningful way such that the same portion of t is visible from each such partition (we call such partition a *slice*). (ii) The visibility of t from such a partitioning should be the average visibility from each point of the slice. However, as the same portion (or a segment) of t is visible from every point of that slice, we can compute that segment's visibility from a limited number of points of the trajectory partition without any loss of accuracy. (iii) As the same segment of t can be visible from multiple slices of a trajectory l, for each segment of t, we find the slice of l that provides the maximum visibility. This value is the visibility for that segment of t from l. (iv) Finally, the summation of the visibility for each segment of t is the total visibility of t from l. We discuss the details of the steps in the following.

5.1.1 Finding the necessary segments of trajectory

A trajectory consists of many segments, but we only need the segments that are inside the visible range. Therefore, we first determine the terminal points (s_i, f_i) of trajectories, and then divide the trajectories into four groups, *up*, *down*, *left*, *right*, based on their relative position from the target.

Visibility of the target is the aggregation of the visibility for the *up*, *down*, *left*, *right* sides of the target. We extend the target line and take the segments inside the region created by the intersection of the target line and visible range. Moreover, if the target line intersects any trajectory segment then the segment is divided into two parts and only the subsegment inside the region is considered.

Let us consider target t_2 and the associated trajectories l_1 and l_2 in Fig. 4. After taking the segments of the trajectories inside the visible range, we have the set of trajectories $L_{t_2} \subset L$ associated with t_2. We need to divide it into four sets of segments $S_{t_2(up)}$, $S_{t_2(down)}$, $S_{t_2(left)}$, $S_{t_2(right)}$, each associated with a specific side of the target.

We now determine the set of segments for each trajectory with respect to the target t_2.

$$\begin{aligned} SS(l_1, t_2) &= \{s_1d, de\} \\ SS(l_2, t_2) &= \{s_2f, fg\} \\ SS(l_3, t_2) &= \{\} \end{aligned}$$

Since l_3 has no segment inside the visibility range of t_2, we need to compute the visibility of t_2 with respect to l_1 and l_2 only, and thus $L_{t_2} = \{l_1, l_2\}$. In Fig. 4, for upper side of the target, we extend the target line X_{up} and only take the segments that are inside the upper side of X_{up} and the visibility range.

$$S_{t_2(up)} = \{s_1d, de\}$$

Using the same process, we can also find the *down*, *left*, and *right* side group of the trajectory segments.

5.1.2 Trajectory partitioning by obstacle projection

To compute the visibility we need to find a continuous portion of the target that is visible from a segment or some portion of it. Thus, the key intuition of trajectory partitioning is to divide the trajectory into different *equi-visible* subsegments (slices) such that the same subsegment of the target is visible from each point of an equi-visible slice.

We first discretize the target by taking equidistant boundary points on each side of the target. Then we take the projection from each boundary point of the target with respect to each obstacle. These projections divide the trajectory segments into different *equi-visible* slices.

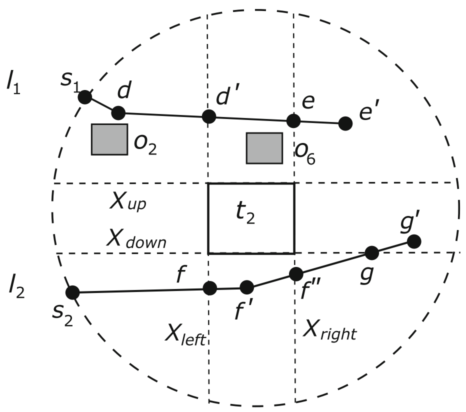

Fig. 4 Segments inside different side groups

From each equi-visible slice of the trajectory, we compute the visibility of a continuous portion of the target, which is the line connecting two or more boundary points that are visible from the slice.

Finally, we compute the visibility of the target from the entire trajectory segment. We summarize the trajectory partitioning process as follows.

Consider the example shown in Fig. 5, where the target object is t_1, and the obstacle is o_1. Two segments ab and bc of t_1 are at the *up* side of the target. Here, we have three boundary points, b_1, b_2, b_3 on the *up* side of target. When there are no obstacles, all the boundary points are visible from both segments. Therefore, we initialize the count of the number of visible points (also referred to as the *visibility count*) for each segment to 3. Now, we take a projection from boundary point b_1 with respect to obstacle o_1. Since ab is completely outside of the projection area, its visibility count does not change. However, the projection for b_1 intersects the segment bc at two points e_1 and e_2, which divides the segment into three slices be_1, e_1e_2, and e_2c; thus we update the visibility count in each slice. In this case, the visibility count of e_1e_2 is now 2. We repeat the above process for

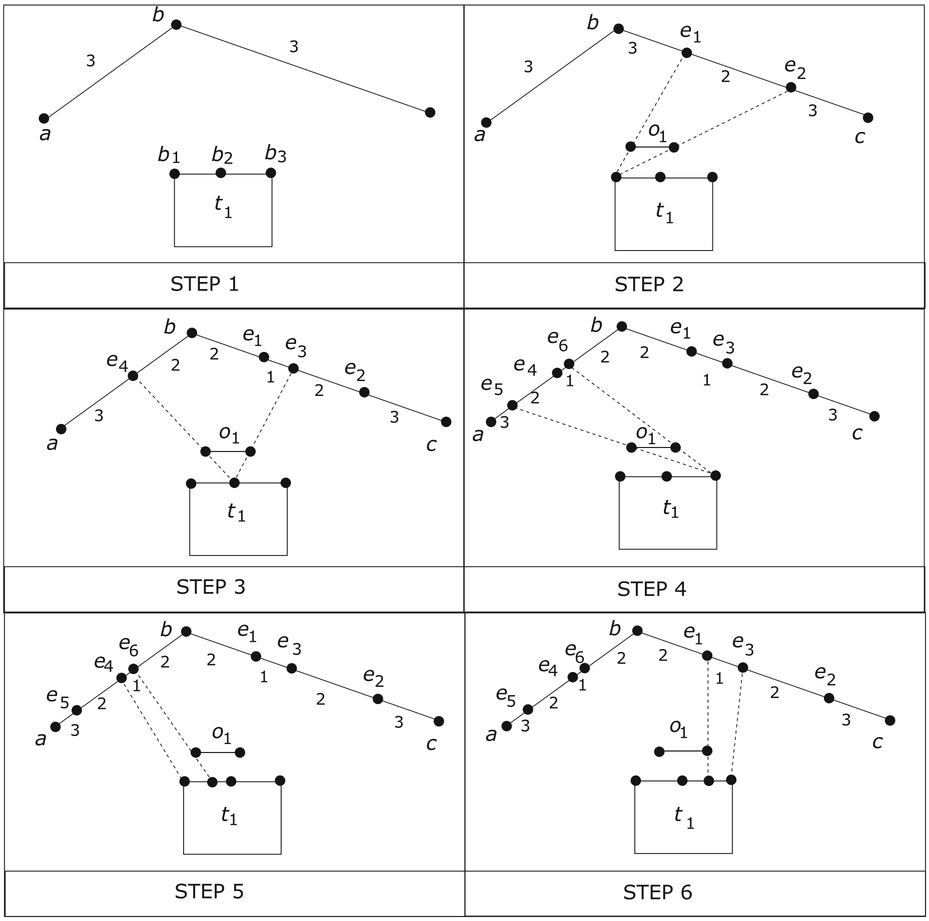

Fig. 5 Projection for a boundary point on trajectory segments are shown in Steps 1-4

the other two boundary points: b_2 and b_3. We get a set $X_{ab} = \{ae_5, e_5e_4, e_4e_6, e_6b\}$ of equi-visible slices for the segment ab, where each slice is associated with visibility count denoting the number of boundary points that can be seen from the slice. Similarly, we get a set $X_{bc} = \{be_1, e_1e_3, e_3e_2, e_2c\}$ of equi-visible slices for the segment bc.

When one equi-visible slice has visibility count greater than 1, we connect those boundary points to have a target line segment, and compute the visibility of the line segment from the slice. However, if an equi-visible slice has a visibility count of exactly 1 – one boundary point is visible from the slice – then we project back from the cell onto the target, and find a target line segment that is visible from the slice. Figure 5 (Step 5) shows an example, where e_4e_6 has a visibility count of 1, and one boundary point, b_1 can be seen from this slice. After projecting back from the slice, we get a target line segment b_1b_1', and compute the visibility of this line segment from e_4e_6.

So far we have computed the visibility of a portion (or segment) of the target from different slices of the trajectory. However, it is possible that the same portion of the target is visible from different slices of the same trajectory. Thus, if we sum up the visibility values of all slices to get the total visibility of the target from the entire trajectory, then the visibility of the same slice can be counted more than once for a trajectory. To avoid this, we first distribute the visibility value or score of a target line segment (w.r.t. a trajectory slice) among its boundary points, and for each trajectory slice we maintain a score list of all boundary points. Then, when we sum up the visibility scores of each slice, we only count the score of a boundary point once for the slice where the boundary point gets the maximum score.

Algorithm 1 *vis_seg*(s, t, ω).

```
Input  : A target $t$, $\omega$ a specific side of $t$, $s$ a trajectory segment
Output: Visibility score list and total visibility score of $t$ with respect to segment $s$ for
        that side
1  $X_s \leftarrow GetSegmentParts(s, t, \omega)$
2  $X_s \leftarrow Sort(X_s)$
3  Initialize computed visibility $V = 0$
4  Initialize score list $\xi_s = [0, 0, ..., 0]$
5  foreach $slicex \in X_s$ do
6      Initialize score list for slice $\xi_x = [0, 0, ..., 0]$
7      $B_x = GetVisiblePoints(x)$
8      $S_t = CreateLinesFromPoints(B_x)$
9      foreach $linet' \in S_t$ do
10         $v = sim\_vis(x, t')$
11         $\xi_x = UpdatePartScoreList(\xi_x, PointsIn(t'), v)$
12     end
13     $\xi_s = UpdateScoreList(\xi_s, \xi_x)$
14 end
15 $V = \sum(\xi_s)$
16 return $(\xi_s, V)$
```

Algorithm *vis_seg*$(s, t, side)$ summarizes the visibility computation of target t for side ω with respect to a trajectory segment s. After taking the projection from each boundary point, the segment s is divided into a set X_s of equi-visible slices. Then we sort X_s according to the number of visible boundary points n_x for slice x. We iterate each slice x in X_s, and initialize the score list of boundary points for the slice, ξ_x as zeros. For each slice x, we then form the line segment t' by connecting all boundary points visible from x, and compute the corresponding visibility $sim\_vis(x, t')$ as defined in Section 4. After that we update score list ξ_x of slice x. The score of i·th boundary point for slice x is defined as follows.

$$\xi_x[i] = \begin{cases} 0, & \textbf{if } b_i \notin t' \text{ , } \textbf{where } t' \in S_t \\ \frac{sim\_vis(x,t')}{w}, & \textbf{if } b_i \in t' \text{ , } \textbf{where } t' \in S_t \end{cases}$$

Here, w is the normalization factor, which is used to equally distribute the visibility score of a target line to its boundary points. When more than one adjacent boundary points are visible from a trajectory slice, w represents the number of boundary points that form the line segment t'. However, when only one isolated boundary point is visible from a trajectory slice, then we calculate w differently. Let $|x|$ be length of the visible portion of the target and $|b|$ be the length of two initial equi-spaced boundary points on the target. Then, the weight, w is defined as $1 + \frac{|x|}{|b|}$, which basically normalizes the visibility score uniformly among the boundary points of the visible portion of the target.

After iterating over all equi-visible slices of the trajectory segment, we compute the score list ξ_s of the segment s, where the score of i·th boundary point for segment s is computed as follows:

$$\xi_s[i] = \max_{x \in X_s}(\xi_x[i]).$$

Finally, the algorithm returns the visibility score V of target t of a given side with respect to segment s as $V = \sum_i \xi_s[i]$. Consider a segment s is divided into a maximum number of n_x slices i.e., $|X_s| = n_x$. Then sorting X_s will take $\mathcal{O}(n_x \log n_x)$. For each slice x, getting visible points and create line from points can be conducted in linear time. However, there will be maximum $\frac{b}{2}$ individual lines that are visible in a specific target side and updating score list will take $\mathcal{O}(n_x)$ time. Since number of boundary points considered in this problem is a constant and $b << n_x$, it will take $\mathcal{O}(n_x)$ to update the score list for each slice in X_s. Therefore, the run time of Algorithm 1 is $\mathcal{O}(n_x \log n_x) + \mathcal{O}(n_x) * \mathcal{O}(n_x) \simeq \mathcal{O}(n_x^2)$.

The steps of the visibility computation for target t_1 w.r.t. trajectory segment ab (Fig. 5) is demonstrated in Table 3. For each slice, we first find the visible boundary points of the target, and compute the visibility of the target line segments. Then, we distribute the visibility score to the boundary points of t_1 according to a normalization factor w, and get the score list for slice x.

In our example, all the three boundary points (b_1, b_2, b_3) are visible from slice ae_5 and the visibility score for target line b_1b_3 is assumed to be 0.75. The normalization factor, w for slice ae_5 will be 3, which is the number of visible boundary points from the slice. Then the score for each visible boundary point is computed by dividing the visibility score, 0.75, by the normalizing factor (w), 3, and the total score list is updated to [0.25, 0.25, 0.25]. The visibility computation for the slice e_5e_4 is done similarly with two adjacent visible boundary points b_1 and b_2.

On the other hand, only one visible boundary point b_1 is visible from slice e_4e_6 (Row 4 of Table 3). In this case, we have to first determine the target line segment b_1b_1', where 0.6 be

Table 3 Visibility computation for a trajectory segment *ab*

Slice	Visible points	Visibility score	w	Score list for slice	Score list for segment
ae_5	b_1, b_2, b_3	0.75	3	[0.25, 0.25, 0.25]	[0.25, 0.25, 0.25]
e_5e_4	b_1, b_2	0.60	2	[0.30, 0.30, 0]	[0.30, 0.30, 0.25]
e_6b	b_1 and b_3	0.40, .20	1.2, 1.1	[0.167, 0, 0.18]	[0.30, 0.30, 0.25]
e_4e_6	b_1	0.60	1.6	[0.375, 0, 0]	[0.375, 0.30, 0.25]

the visibility of score of the target line from e_4e_6. The normalization factor w is computed as 1.6, and the visibility score of b_1 is updated as $0.6/1.6 = 0.375$. Similarly, two isolated boundary points b_1 and b_3 are visible from slice e_6b. So, we have to first determine the visibility for two separate line segments, for both target lines containing b_1 and b_3. Here, 0.4 and 0.2 are the visibility scores for these two segments, respectively. The normalization factors for these target lines are computed as 1.2 and 1.1, respectively. Therefore, scores for b_1 and b_3 are updated as $0.4/1.2 = 0.167$ and $0.2/1.1 = 0.18$, respectively and the visibility score for b_2 is updated to 0 as it is not visible from slice e_6b.

Finally, we update the score list for segment ab (last column of Table 3), where each boundary point gets the maximum score from score list of all slices. At first, the score list for trajectory segment ab is initialized as $\xi_{ab} = [0, 0, 0]$. After calculating the score list ($[0.25, 0.25, 0.25]$) for slice ae_5, ξ_{ab} is updated as $[0.25, 0.25, 0.25]$. For slice e_5e_4, the score list is $[0.30, 0.30, 0]$. Thus, the visibility scores for b_1 and b_2 are only updated, and the new visibility score list for the segment is $\xi_{ab} = [0.30, 0.30, 0.25]$. We continue this process for the rest of the slices, and the final score list for trajectory segment ab is $\xi_{ab} = [0.375, 0.30, 0.25]$. Therefore, the visibility score for target t_1 w.r.t trajectory segment ab is defined as $V = 0.925$.

To compute the visibility score of the entire trajectory l, and for a specific side ω, we define the score of the i·th boundary point for trajectory l as the maximum of all segments of that trajectory which are inside in that specific side group.

$$\xi_l[i] = \max_{s \in SS(l,t) \cap S_{t(\omega)}} (\xi_s[i]).$$

Thus, the visibility score of trajectory l for side ω can be computed as $l.visibility[\omega] = \sum_i (\xi_l[i])$. Finally, the total visibility of t with respect to l is the aggregation of visibility for all sides in t, which is

$$vis(l, t) = \sum_{\omega} l.visibility[\omega]$$

In our example, we can compute the visibility of target t_1 w.r.t the trajectory segment bc using the same process as ab. Let the computed score list for bc be $\xi_{bc} = [0.27, 0.40, 0.35]$, and the score list for ab be $\xi_{ab} = [0.375, 0.30, 0.25]$ as demonstrated in Table 3. Then, the score list w.r.t. to the entire trajectory l_1 is defined as $\xi_{l_1} = [0.375, 0.40, 0.35]$, and the total visibility value of the upper side is $l_1.visibility[up] = 1.125$.

5.2 A straightforward approach

In case of multiple targets, i) this approach processes each target sequentially, ii) retrieves all trajectories and obstacles inside the visible range of the target, iii) computes the obstructed visibility of the target with respect to the trajectories, and iv) outputs the top-k trajectories combining all targets as a result of kAMVT query.

Although this method provides us the desired solution, it considers all trajectories and obstacles within the visible range. Thus the straightforward approach leads to several complexities: i) The computational cost will be excessive. ii) Many trajectories and obstacles present inside the visibility range have no significant effect on the visibility of the targets. Therefore, considering all of them will be unnecessarily expensive. iii) the computational cost will increase as the number of targets increases.

5.3 Proposed approach

In order to decrease the computational cost, we propose an advanced approach that includes some efficient pruning mechanisms which greatly reduce the number of obstacles and trajectories for both single and multiple targets. The steps of our proposed approach are explained in the following subsections.

5.3.1 Determining a reduced obstacle set

A major challenge in our problem is to manage the huge obstacle set associated with each target object. Thus, we need a method to prune out the obstacles which do not affect the visibility calculation, and find a reduced obstacles set O_t for target t where, $O_t \subset O$.

The obstacles are retrieved from an R-tree in increasing order of distance from the target, since we assume that closer obstacles have a higher chance of blocking the view. Moreover, an obstacle outside of the region covering all trajectories can not block the view of target. For each target, a *Trajectory Segmentation Range* is defined as the MBR with a minimum extent that encloses all trajectories inside the visible range of the target.

At first, we initialize the reduced obstacle set $O_t = \emptyset$ for target t. Then, we start from the obstacle which is nearest to the target, and continue up to the *Trajectory Segmentation Range* by increasing the distance of the obstacle from the target. The main intuition is that, an obstacle, or a whole MBR shadowed by other obstacles retrieved already has no effect on target. Therefore an *Invisible Range* is maintained. We define the *Invisible Range* of an obstacle as the area inside the *Trajectory Segmentation Range* which is shadowed by taking the projection from the target to that obstacle. Initially *Invisible Range* is empty. Whenever a new obstacle is inserted into O_t, we take projection of the obstacle from the target and the *Invisible Range* is updated. The following lemma is used to prune out an obstacle that does not need to be retrieved from the R-tree.

Lemma 1 *Let o be an obstacle in the Trajectory Segmentation Range of a target t. The obstacle o can be safely pruned if it is completely inside the Invisible Range, taken from t to all obstacles o' that have been retrieved so far, $\forall o' \in O_t$. If a whole MBR in inside the Invisible Range, then all obstacles inside it can be safely pruned.*

While calculating visibility for a specific side of the target, we need to find a projection for the obstacles that are between that target side and the *Trajectory Segmentation Range*. Therefore for each side ω, we need to find a subset of obstacles $O_{t,\omega}$ from O_t, where each obstacle $o \in O_{t,\omega}$ are associated with that side.

An example scenario is illustrated in Fig. 6 with the target t_1 and segments ab, bc, cf_1. The provided obstacle set here consists of five obstacles, $O = \{o_1, o_2, \ldots, o_5\}$. For simplicity, only the upper side of the target t_1 is considered.

The first retrieved obstacle here is o_1 since it is the closest one to t_1. Then o_2 is retrieved and inserted in O_{t_1}. The next obstacle o_4 is not retrieved because the position of o_4 is left of the leftmost extreme point a, and therefore outside of *Trajectory Segmentation Range*. The next obstacle o_3, however, is completely shadowed by the obstacle o_1, and is therefore not retrieved. The obstacle o_5 follows, and is above the uppermost point b and so need not be retrieved. As a result, the reduced obstacle set obtained here is $O_{t_1} = \{o_1, o_2\}$.

An obstacle cannot effect the visibility of the segment if it is completely outside the visibility region of the target with respect to that segment. This visibility region is formed by joining the end points of the segment and the target side. In the reduced obstacle set, o_1

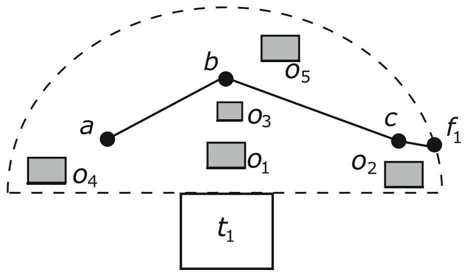
Fig. 6 Obstacle retrieval for a target

is inside the visible region of *ab* but o_2 is outside. Thus, we need to calculate the visibility of *ab* only with respect to obstacle o_1.

5.3.2 Trajectory pruning for a target

In this section we demonstrate the process of trajectory pruning with respect to a single target. In order to facilitate pruning, we also maintain an upper and lower bound on the visibility of individual trajectories to compare these values with a global lower bound.

Upper bound: The upper bound on the visibility of a trajectory l is the maximum visibility that l can provide, denoted by $l.ub$. For each trajectory, the simplified visibility without obstacles is calculated, which is the initial upper bound. For a trajectory l, the initial upper bound is $l.ub = sim\_vis(l, t)$ and l is considered as a geometric line inside the visible region of t.

Lower bound: The lower bound on the visibility of a trajectory l is the minimum visibility the trajectory can provide, denoted by $l.lb$. The lower bound is initially zero, which indicates the worst case scenario. If visibility has been computed for a portion of a trajectory in the presence of obstacles, then this is the lower bound on visibility. We assume in the worst case scenario that the target will not be visible at all from the remaining portion of the trajectory.

We can consider the upper bound as the aggregation of the visibility for the computed slice and the simplified visibility for the remaining slice. In the best case scenario, there will be no obstacles, and the visibility obtained is equal to the simplified visibility. It is evident that, with each iteration, the upper bound will decrease, and the lower bound will increase. The upper bound can be updated according to the following lemma.

Lemma 2 *Let l be a trajectory. If the visibility has been computed for a portion x in l with respect to a target t, and x' is the remaining slice, then $l.ub = vis(x, t) + sim\_vis(x', t)$.*

In Lemma 2, the update of $l.ub$ can be simplified by another equation using algebraic manipulation since the former will be computationally intensive if the number of segments associated with the slice x' is huge. Let $l.ub'$ be the previously computed upper bound of l and we calculate visibility for portion x of l. Then we can derive the new upper bound $l.ub$

by simply taking the difference between the simplified visibility and actual visibility of x from the previous upper bound. Then the update equation becomes,

$$l.ub = l.ub' - (sim\_vis(x,t) - vis(x,t))$$

The lower bound can be updated using to the following lemma.

Lemma 3 *Let l be a trajectory. If the visibility has been computed for a slice x in l with respect to a target t, then $l.lb = vis(x,t)$.*

Now, we can use this upper and lower bound to prune trajectories, by maintaining a global lower bound LB. It is defined as the lower bound on the visibility among the top-k trajectories inside the visibility range of t. The global lower bound is also initialized as zero and updated accordingly. For any trajectory, if the upper bound is smaller than the global lower bound, then it can be pruned. This indicates that the trajectory, even in the best-case scenario, will provide worse visibility than the trajectories those are already in top-k list.

Lemma 4 *Let t be a target t and l be a trajectory. If LB is the global lower bound with respect to t, then l can be pruned if the following condition holds: $l.ub < LB$.*

Consider the example illustrated in Fig. 7, where we consider the downward direction from the target. Here, t_1 is the target object, and l_2 is the trajectory being considered, which has three segments s_1, s_2 and s_3, and the trajectory l_3 has two segments s_4 and s_5.

Initially, the upper and lower bound for l_2 are determined. Then the visibility for the the segments s_1, s_2, and s_3 are computed in that order, according to the distance from the target. Now, the upper bound $l_2.ub$, lower bound $l_2.lb$ and global lower bound LB change according to Lemma 2 and Lemma 3.

Now, segment s_4 of l_3 is considered. After defining the upper and lower bounds of l_3, we check the pruning condition for s_4 according to Lemma 4. If the condition holds, we can safely conclude that even the best-case visibility provided by l_2 will be worse than the trajectories we have considered beforehand. Thus, l_3 can be pruned. Otherwise, the computation continues.

5.3.3 Processing *k*AMVT query for single target

Algorithm 2 takes a specific target t, its associated trajectory set L_t, and the obstacle set O and returns a list R containing top-k trajectories with maximum visibility for target t.

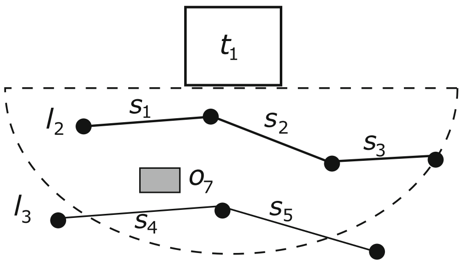

Fig. 7 Pruning trajectories based on upper and lower bounds

Algorithm 2 kMVT(k, t, L_t, O).

```
Input  : Items returned, $k$, target object $t$, a trajectory set $L_t$ associated with $t$, an
         obstacle set $O$
Output: A list $R$ of $k$ trajectories in decreasing order of visibility
1  Initialize $R = \emptyset, LB = 0$
2  foreach side $\omega$ do
3      $S_\omega = DivideSegments(t, L_t, \omega)$
4  end
5  foreach Trajectory $l \in L_t$ do
6      Initialize $l.lb = l.total\_vis = 0$, $l.ub$ and $l.visibility[\omega] = 0$ for each side $\omega$
7      Initialize $\xi_l = \{0, 0, \ldots, 0\}$ for each side $\omega$.
8  end
9  $O_t = Retrieve\_Reduced\_Obstacle\_Set(t)$
10 foreach side $\omega \in \{up, left, down, right\}$ do
11     $O_{t,\omega} = Get\_Specific\_Side\_Obstacles(O_t, \omega)$
12     foreach Segment $s \in S_\omega$ do
13         $Trajectory l = s.parent$
14         if $l.ub < LB$ then
15             continue
16         end
17         foreach Obstacle $o \in O_{t,\omega}$ do
18             if $o$ is outside of visible region of $s$ then
19                 continue
20             end
21             foreach boundary point $b$ of target line do
22                 $s = Modify\_Segment(s, b)$
23             end
24         end
25         $(\xi_s, V_s) = vis\_seg(s, t, \omega)$
26         Update $l.lb$, $l.ub$ and $LB$
27         $\xi_l = UpdateScoreList(\xi_l, \xi_s)$
28         $l.visibility[\omega] = \sum(\xi_l)$
29         $UpdateVisibility(l)$ and $Insert(R, l)$
30     end
31 end
32 Return first $k$ trajectories from $R$
```

Here, we maintain the priority list of trajectories R, in decreasing order of visibility and a global lower bound LB. At first L_t is divided into four groups associated with each side (Lines 2-4). Each S_ω contains segments associated with that side sorted according to their distance from that side. Then the upper bound $l.ub$, lower bound $l.lb$, and the total visibility of each trajectory associated with the target *total_vis* are initialized. The visibility score and the visibility score list of the target boundary points for each side of the target are initialized, for every trajectory (Lines 5-8). Finally a reduced obstacle set O_t is retrieved from the R-tree as described in Section 5.3.1 (Line 9).

For each side ω of the target, we take a subset of obstacles $O_{t,\omega}$ that are associated with that side (Line 11). Then for each segment s, we check the pruning condition according to Lemma 4. If this condition holds, this trajectory can be pruned (Lines 14-16). Otherwise we iterate over all of the obstacles in $O_{t,\omega}$, and check if an obstacle affects the visibility of the segment. If not, we can ignore the obstacle (Lines 18-20). Otherwise, we calculate the effect of projection of that segment for each boundary point according to Section 5.1.2 (Lines 21-23).

After iterating all obstacles for a segment, we compute the visibility score list ξ_s, and also the total visibility score V_s for the segment, according to Algorithm 1 (Line 25). The upper bound, lower bound, and LB are updated according to Lemma 2 and Lemma 3. We also update the visibility score list for the trajectory, ξ_l, as mentioned in Section 5.1.2 (Line 27). We then use ξ_l to compute the visibility score of the trajectory for the side ω (Line 28). Next, the overall visibility of the trajectory is updated, and the trajectory is inserted into the list R accordingly. If the trajectory is already in R, its visibility value is updated and its position in the list changes (Line 29). Finally, we return the top-k trajectories from R as the desired output (Line 32).

The run time of Algorithm 2 in worst case would be similar to the straightforward approach with no pruning method. But all these efficient pruning mechanism significantly reduce the run time in real scenario that we discuss in Section 6.

Proof of correctness We now show that Algorithm 2 finds the list of top-k trajectories with respect to a given target. For this reason we have to prove following two statements.

(i) First, k trajectories in R provide better visibility than rest of the trajectories in R.
(ii) Whenever a trajectory is pruned, the visibility will be less than the visibility of the k·th trajectory in R.

The output list R is a priority list based on the value of the visibility score, $vis(l, t)$. Here $vis(l, t)$ is the visibility of the target t with respect to trajectory l, considering all sides of t. So $vis(l, t) = \sum_{\omega} l.visibility[\omega]$, as defined in Section 5.1.2. As a result, whenever a trajectory is inserted into R, it will not disrupt the order of visibility. So statement (i) remains valid for any scenario.

For the second statement, the global lower bound LB is updated in each iteration, where LB is the value of lower bound of k·th trajectory in R. Whenever the iteration is over, $vis(l, t)$ of a trajectory l will be equal to its lower bound $l.lb$. Moreover, the upper bound of the trajectory signifies the maximum visibility that the trajectory can provide. So at each iteration, we have the following fact for any trajectory l:

$$l.lb \leq vis(l, t) \leq l.ub$$

Let l be a trajectory that is being pruned, and l' is the k·th trajectory in R. So the global lower bound $LB = l'.lb$. According to the pruning condition,

$$\begin{aligned} & l.ub < l'.lb \\ \Rightarrow vis(l, t) \leq\ & l.ub < l'.lb \leq vis(l', t) \\ & \Rightarrow vis(l, t) < vis(l', t) \end{aligned}$$

It is evident that l will not provide better visibility than the k·th trajectory that has been achieved already. As a result, the statement (ii) is indeed true. Therefore our proposed solution will return the top-k trajectories with best visibility for the given target.

5.4 Visibility computation for multiple targets

In this Section, we provide an efficient mechanism for visibility computation considering multiple targets. We use total visibility $V(T, l)$ as the scoring function of each trajectory l which was defined in Section 3. Evaluating this score for all trajectories while finding the top-k, would incur huge computational costs. Therefore, we propose additional optimization techniques for multiple targets, and a target ordering technique. This helps prune away many trajectories, and in some cases, a target instantly. Finally, we provide an algorithm for the kAMVT query.

5.4.1 Target ordering technique

In the initial step, the target which has the highest number of trajectories inside its visible range is selected.

After the computation is done for initial target, we pick the first trajectory in the top-k list found so far, and consider the target which has l inside the visible range. If there are multiple such targets, the one which has the highest number of trajectories inside its visible range is picked.

As k trajectories are required, preference is given to targets which already have higher ranked trajectories inside the visible range.

If all targets that have l inside their visible range have been considered, the next trajectory in the top-k list is processed. If no such trajectory can be found in the top-k, then we choose the unevaluated target that have a higher number of trajectories, or trajectories with an aggregated higher weights (preferences).

5.4.2 Trajectory pruning for multiple targets

For each trajectory, an expected upper bound on its visibility is maintained to facilitate pruning.

Expected upper bound: The expected upper bound of a trajectory l is defined as the maximum visibility that l can provide, considering every target, and is denoted by $l.expected$. We initialize the expected upper bound with $|T| \times max\_vis$, where $|T|$ is the total number of targets in T and max_vis is a predefined maximum visibility of a target with respect to a trajectory. Updates can be managed using the following lemma.

Lemma 5 *Let l be a trajectory, if $l.vis$ is the total visibility computed so far for n' targets in T, then the expected upper bound of l, $l.expected = l.vis + (|T| - n') \times max\_vis$.*

For each trajectory l, we compare its expected visibility with the minimum total visibility and prune if needed.

Minimum total visibility: The minimum total visibility, denoted by min_total_vis, is defined as the total visibility $l.vis$ of the k·th top trajectory l that has been found so far.

If the expected visibility of l is less than minimum total visibility, then l will not disrupt the ranking of top-k list found so far, and can be pruned. The pruning condition for trajectories is as follow.

Lemma 6 *Given a trajectory l, if l.expected is the expected maximum visibility of l , and min_total_vis is the minimum total visibility, then l can be pruned if the following condition holds:* $l.expected < min\_total\_vis$.

5.4.3 Target pruning

If Lemma 6 holds for all trajectories inside the visibility range of t, then t can be eliminated from computation altogether. This signifies that no trajectory inside the visible range of t can ever surpass the visibility of the k·th top trajectory found so far.

Let us consider the example scenario illustrated in Fig. 8. Here, the targets are t_1 and t_2, the trajectories are l_1, l_2, and l_3 and the obstacles are $o_1, o_2, \ldots, o_7$. We want to find the top most trajectory ($k = 1$) with maximum visibility for all targets. Now, the trajectories inside the visible range of the targets are, $L_{t_1} = \{l_1, l_2, l_3\}$ and $L_{t_2} = \{l_1, l_2\}$

In the example, we maintain a priority list of targets $T' = \{t_1, t_2\}$, since t_1 has the maximum number of trajectories inside the visibility range. Therefore, Algorithm 2 is processed for t_1 at first and the result is $\{l_2\}$ since the trajectory l_2 is less obstructed. This is inserted into the resultant list of top-k trajectories R, and thus, $R = \{l_2\}$, and t_1 is removed from T'.

Now, while choosing the next target, it can be seen that l_2 has the highest visibility, and thus we consider the other target that has l_2 in its visible range and is higher in the priority list T', so we choose t_2 next, and remove it from T'. Since $L_{t_2} = \{l_1, l_2\}$, if the expected upper bound of l_1 is less than *min_total_vis*, then we can simply prune out l_1. Thus, for target t_2, the returned trajectory will be l_2 and we get our resultant list $R = \{l_2\}$ as the result of the kAMVT query.

5.4.4 Selecting candidate trajectories w.r.t. a target

In this section, we propose a heuristic to determine the number of trajectories that needs to be accessed w.r.t. individual target, which are potential candidates for the top-k list for the kAMVT query. Later, we present a Lemma that defines an upper bound visibility score based on the already retrieved trajectories for checking the eligibility of remaining trajectories.

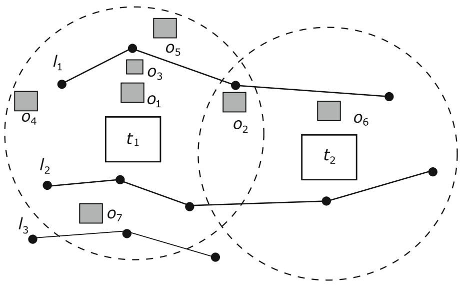

Fig. 8 Visibility measurement for all targets

Accessing top-k trajectories for each target and then combine the top-k lists of all targets do no guarantee the top-k results for the kAMVT query. One possible approach can be to access top k'' trajectories, where $k'' >> k$, for each target and then combine these trajectories to get the final top-k. This approach results in un-necessary accessing of trajectories that cannot be a part of the answer. Thus, we provide a heuristic to determine the number of trajectories, k' that needs to be accessed for selecting initial set of candidate trajectories w.r.t. each target.

We first explain the concept using an example as shown in Fig. 9. In this figure, $l.vis$ is the visibility score for trajectory l that has been computed so far, $l.expected$ is the expected upper bound of l that can be achieved. A trajectory l can be included in top-k list if its visibility score $l.vis$ exceeds min_total_vis. Let L_t be the set of trajectories that are inside the visible range of t and cannot be pruned according to Lemma 6. $\overline{l.expected}$ and $\overline{l.vis}$ are the average expected upper bound and the average so far computed visibility score of all trajectories in L_t, respectively. Let us assume that k' be number of trajectories that may be the potential candidates for the final top-k list of the kAMVT query. Thus, according to the unitary method, for target t, we define k' as follows.

$$k' = \frac{\overline{l.expected} - min\_total\_vis}{\overline{l.expected} - \overline{l.vis}} \times |L_t|$$

For the first few targets, the value of k' will be large, as initially the value of min_total_vis will be small and consequently the difference between $l.expected$ and min_total_vis will be large. As we retrieve more trajectories for successive targets, the value of min_total_vis will increase, and also the value of $l.expected$ will get closer to the actual visibility score $l.vis$, and thereby the value of k' will be small.

However, the above heuristic for k' does not guarantee that we access all candidate trajectories for the final top-k list of the kAMVT query. To ensure that all candidate trajectories have been accessed to confirm the correctness of the top-k answer, we provide the following lemma. According to the above heuristic, let $k'_1, k'_2, \ldots k'_{|T|}$ be the number of trajectories retrieved for targets $t_1, t_2, \ldots t_{|T|}$, respectively. Let $l_{k'}$ be the k'th trajectory of the top-k' list of target t, and $vis(l_{k'}, t)$ be the corresponding visibility score of the trajectory. Then, we can get the upper bound visibility score as $UB = \sum_{t \in T} vis(l_{k'}, t)$. An unexplored trajectory l can still be the candidate for the top-k if the expected score, $l.expected$ of l is greater than the upper bound score, UB. Thus, we have the following lemma.

Lemma 7 *Given a target t and a set of trajectories L_t inside the visible range of t, where $\forall l \in L_t, l.expected < min\_total\_vis$. Let R_t be the list of top-k' trajectories w.r.t. target t. Then, a trajectory $l \in L_t \backslash R_t$ can be a candidate if $l.expected > UB$.*

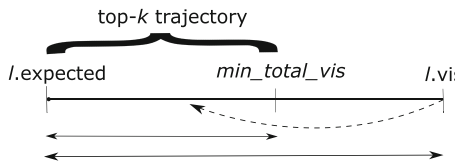

Fig. 9 Visibility Range for a trajectory

5.4.5 Processing the kAMVT query

Algorithm 3 kAMVT query.

```
Input  : Number of results, $k$, target object set $T$, and a Preference vector of targets $P$
Output: A list of $k$ trajectories in decreasing order of visibility
1  Initialize priority list of target $T' = \varnothing$
2  foreach $Target\ t \in T$ do
3      $L_t = RetrieveTrajectory(t)$
4      $t.preference = L_t.size() * p_t$
5      $T'.insert(t)$
6  end
7  Initialize $R = \varnothing$, $min\_total\_vis = 0$ and $UB = 0$
8  while $T' \neq \varnothing$ do
9      Target $t = SelectAppropriateTarget(T')$
10     $T'.remove(t)$
11     foreach $Trajectory\ l \in L_t$ do
12         if $l.expected < min\_total\_vis$ then
13             $L_t.remove(l)$
14         end
15     end
16     if $L_t.size() = 0$ then
17         continue
18     end
19     $k' = SelectTrajectory(L_t, min\_total\_vis)$
20     List $R_t = k\text{MVT}(k', t, L_t, O)$
21     $UB = UB + vis(R_t[k'], t)$
22     foreach $Trajectory\ l \in R_t$ do
23         $l.vis = l.vis + vis(l, t) * P(t)$
24         Update $l.expected$
25         $Insert(R, l)$
26     end
27     $min\_total\_vis = R[k].vis$
28 end
29 foreach $Target\ t \in T$ do
30     foreach $Trajectory\ l \in L_t$ do
31         if $l \notin R_t$ and $l.expected > UB$ then
32             $CaculateVisibility(l, t)$
33             $Insert(R, l)$
34         end
35     end
36 end
37 Return top-$k$ trajectories from $R$
```

This algorithm takes an integer k, a target list T and associated preference vector P, total obstacle set O and trajectory dataset L as input. The top-k trajectories that provide the best visibility considering all targets in T are returned.

In order to find the top-k trajectories for multiple targets, two priority lists R and T' of trajectories and targets respectively are maintained. The trajectories are inserted into R according to their total visibility $l.vis$ computed so far. After iterating for all targets, the top-k trajectories from R will have the highest visibility score $V(T, l)$.

At first, we initialize T' (Line-1) which is basically an open set of targets whose visibility have not been evaluated yet. For each target t, the associated trajectory list L_t inside the visible range are retrieved. The targets are inserted into T' as a function of their preference level, and the number of trajectories inside their visible range (Lines 4-5). Then, the priority list of trajectories R and the global bound *min_total_vis* are initialized (Line 7).

The evaluation process is continued until T' is empty. We select the next target based on the process mentioned in Section 5.4.1 (Line 9), and then the target from the open set T' is removed (Line 10). After getting a target t, for each trajectory l, we check the pruning condition in Lemma 5.6. If the condition holds, it can be removed from L_t (Lines 11-15). If all trajectories inside the visible range of a target are pruned, and the target can be pruned (Lines 16-18). Moreover, we determine the number of trajectories that should be retrieved w.r.t. target t according to the heuristic defined in Section 5.4.4 (Line 19).

After that, Algorithm 2 is processed for target t with the related trajectory set L_t. This query returns a trajectory list R_t containing the top-k' trajectories for target t (Line 20) and the value of upper bound visibility score is updated (Line 21). For each trajectory l in R_t, $l.vis$ and $l.expected$ are updated, and inserted l into R (Lines 22-26). The minimum total visibility is also updated (Line 27).

Next, for each target t, we compare the expected upper bound of any unevaluated trajectory $l \in L_t$ with UB (Lemma 7). If the condition holds, then we calculate its visibility score $vis(l, t)$ for target t and insert into R accordingly (Lines 31-34).

Finally, the top-k trajectories are returned from R as the result of the kAMVT query (Line 37).

5.5 Extending the solution for weighted trajectories

In this section, we describe how the steps of the solution can be extended when each trajectory is associated with a weight denoting its preference. We extend the solution for both cases where the weights can be easily precomputed, or the weights needs to be computed during query time.

Precomputed weight If the weight of each trajectory can be precomputed easily (e.g., weight based on the length of the trajectory), we can extend Algorithm 2 to process the weighted version of the query. However, we have to consider both the visibility of the trajectory $vis(l, t)$, and its weight w_l. Therefore, we maintain a priority list R of trajectories, in decreasing order of $g(vis(l, t), w_l)$, as mentioned in Section 3. Moreover, while updating $l.ub$ and $l.lb$ we need to replace $vis(x, t)$ by $g(vis(x, t), w_l)$, where x is the portion of the trajectory l that has been processed so far.

Therefore, the top-k trajectories of R will have the highest visibility of target t when considering trajectory weights. We can now use Algorithm 3 combined with this variation of Algorithm 2 for each target, to obtain the solution for the kAMVT query.

Computing weights at query time If the weight of each trajectory is a distance for a query specific location – the weight needs to be computed on-the-fly. This can be accomplished by maintaining a priority list R of trajectories in decreasing order of $h(l.vis, \psi_l)$, as mentioned in Section 3. Here $l.vis$ is the total visibility of trajectory l, and ψ_l is its corresponding weight, based on the distance of the starting point of l from the query user. Algorithm 2 returns the list R_t containing the top-k trajectories for a single target t. Then, we have to find the corresponding weight ψ_l for each trajectory $l \in R$ if it has not been computed before. After updating the total visibility $l.vis$ for l, it is inserted into the list R according to its value $h(l.vis, \psi_l)$. Moreover, while updating $l.expected$ and min_total_vis, we have to use $h(l.vis, \psi_l)$ instead of $l.vis$. Therefore, the first k trajectories of R will have the highest visibility when considering their distance from the query.

6 Experimental evaluation

In this section, we evaluate the performance of our proposed algorithms for kAMVT queries with three real datasets: Boston, New York, and Los Angles, and one large scale synthetic data set. As there is no prior work for this problem, so we compare our solutions with a simple baseline. Specifically, we compare the performance among the following four methods: (i) the straightforward approach presented in Section 5.2 as the baseline approach with no pruning (*NP*), (ii) an obstacle pruning *(OP)* based approach that employs obstacle pruning techniques described in Section 5.3.1, (iii) a trajectory pruning *(TP)* based approach that employs the trajectory pruning techniques for single and multiple targets described in Sections 5.3.2 and 5.4.2 respectively, and finally, (iv) the combination of both trajectory and obstacle pruning along with a multiple target based optimizations approach (denoted as *TP+OP*), which acts as our solution.

6.1 Experimental setup

We use Java to implement our algorithms. Experimental evaluations were conducted on a machine with a PowerEdge R820 rack server with 6-core Intel Xeon E5-4600 processors, and 64 GB RAM, and LINUX OS installed. We have used real obstacle and trajectory datasets for our experiments.

6.1.1 Datasets

We have used three real obstacle datasets: New York (NY), Los Angles (LA), and Boston (BN), representing the building layouts of three different cities. A schematic view of these obstacle data are shown in Fig. 10. We have used Foursquare check-ins, taxi trips and bus

(a) Boston (b) New York (c) Los Angeles

Fig. 10 Real dataset obstacle schematic

routes as trajectory data. Obstacle and trajectory datasets are summarized in Tables 4 and 5, respectively. In all experiments, we have normalized the data space into 10,000 × 10,000 span.

New York Datasets: We have used PLUTO (Extensive land use and geographic data at the tax lot level) dataset from the Department of City Planning (DCP) as our NY obstacle dataset. We have used the city buildings of five boroughs of New York: Manhattan, Brooklyn, Queens, The Bronx and Staten Island, which provide a total of 85,8371 obstacles. The PLUTO files contain more than seventy fields derived from data maintained by city agencies. From these attributes, we have used x-coordinate, y-coordinate, and the area of the obstacle to generate x and y extent of each obstacle through interpolation.

We have used the Foursquare check-ins and taxi trips of NY city as our trajectory dataset in this case. For Foursquare check-ins, we have collected the check-ins of each user in each day and considered these as trajectories. For taxi trips, we have only source and destination locations of the trips record in September, 2015 from NYC Taxi and Limousine Commission (TLC). For each start and end locations of the trip, we have used Open Street Map to find the shortest route, and consider this as a trajectory. In total, we have 830,020 trajectories, which include both check-ins and taxi trips, for the NY trajectory dataset.

Los Angles Datasets: We have used building outlines from LAR-IAC2 (2008) of the City of Los Angles (Countywide Building Outlines) as our LA obstacle dataset. The building outlines is represented as a geometric layer file. We have extracted the coordinates (latitude, longitude) of each building, then found the bounding box of the coordinates to get the extents of each obstacle. There are 2,999,997 (almost 3 million) obstacles in our LA dataset.

For trajectories, we have used Los Angeles bus routes and passenger transitions dataset from [43]. There were only 1,209 bus routes in this dataset. We have used the path between two stoppages in a route as a trajectory. Moreover, the passenger transition dataset only contains the start and end locations. We have used the Open Street Map to find a route between every start and end locations pair and consider the route as a trajectory. This produces 177,595 trajectories in total for our LA trajectory dataset.

Boston Datasets: The Boston obstacle set represents 130,043 building objects in the Boston downtown area. In the dataset, objects are represented as 3D rectangles which are used as obstacles in our experiments. In our experiments, we only consider 2D extents of these obstacles (considering the z-axis value as zero) in our experiments. Since, we have not found any real trajectory datasets for Boston, we have generated synthetic trajectories in this dataset. To generate trajectories, first we have generated uniformly distributed coordinate points in the sample space so that no line segment joining the consecutive points intersect any existing obstacle. Each trajectory is composed of 20-30 segments

Table 4 Obstacle datasets

Dataset	# of Obstacles	Description
New York (NY)	858,371	PLUTO (extensive land use and geographic data)
Los Angles (LA)	2,999,997	LAR-IAC2 (2008) of City of Los Angles (building outlines)
Boston (BN)	130,043	Boston downtown from Boston development authority

Table 5 Trajectory datasets

Dataset	# of Trajectories	Description
New York (NY)	830,020	Foursquare Check-ins and NY Taxi trajectory
Los Angles (LA)	177,595	Bus routes and passenger transitions
Boston (BN)	1,000,000 - 3,000,000	Synthetically generated

and each segment is 10-20 units long. We have generated a maximum of three million trajectories in the data space.

Synthetic Dataset: To show the scalability of our approaches, we have also generated large synthetic datasets for both obstacles and trajectories of various sizes. To generate obstacles, we generate the center points of the obstacles using a uniform distribution, and then also generate the extent of each obstacle in the range of 5 to 10 units using the same uniform distribution. In total, we generate four datasets of sizes, 0.5M 1M, 2M, and 3M of obstacles. To generate trajectories, we have first generated uniformly distributed coordinate points in the sample space so that no line segment joining the consecutive points intersect any existing obstacle. Each trajectory is composed of 20-30 segments, and each segment is 10-20 units long. We have generated four datasets of sizes, 0.5M 1M, 2M, and 3M of trajectories.

Query (Target) Datasets: To generate targets, we first choose a rectangular query area as a percentage of the total data space in which targets or query objects can reside. Then we choose targets randomly from the obstacle set inside the query area.

6.2 Performance evaluation and parameterization

We studied the efficiency and scalability of our proposed approaches by varying several parameters. The list of parameters with their ranges and default values in bold are shown in Table 6. In particular, we vary the values of k, number of obstacles n_O, number of trajectories n_L, number of targets n_T, and the percentage of target area A_T. For all experiments, a single parameter is varied while keeping the rest as the default settings.

We have used two $R*$-trees to index obstacles and minimum bounding rectangles (MBRs) of trajectories, where in both case the page size is fixed at 1KB.

For efficiency and scalability, we studied the impact of each parameter on (i) the processing time (run time), (ii) the average number of obstacles retrieved (the number of obstacles that were retrieved from R-tree and considered in the visibility calculation), and (iii) the average number of trajectories retrieved (the number of trajectories that were considered in

Table 6 Parameters

Parameter	Range	Default
k	1, 2, 4, 8, 16	4
Number of targets, n_T	1, 5, 10, 15, 20	10
Percentage of target area, A_T	5, 10, 15, 20, 25	10
Number of trajectories, n_L	500,000, 1,000,000, 2,000,000, 3,000,000	1,000,000
Number of obstacles, n_O	5,000,000, 1,000,000, 2,000,000, 3,000,000	1,000,000
Dataset	Synthetic, Real	Real

visibility calculation), for answering the *k*AMVT query. We run the experiment ten times for each configuration and report the average performance. We consider equal preference for each target in these experiments. For aggregation function, we have used summation of the visibility of each target.

6.3 Experimental results

In this section, we present our experimental results on different datasets, and show the effect of different parameters on the performance of competitive methods.

6.3.1 Effect of k

In this set of experiments, we vary the value of *k* as 1, 2, 4, 8, and 16, and compare the performance of different methods in terms of processing time, and obstacle and trajectory retrieval costs.

New York (NY): Figure 11 shows the effect of *k* for the New York dataset. As *k* increases, the processing time is nearly constant for all algorithms. However, the performance of TP+OP is about 19.5 times faster than NP, 12 times faster than TP, and 3.8 times faster than OP. The number of obstacles and trajectories retrieved for TP+OP is nearly constant for values of *k*. The TP+OP approach retrieves 1.8 and 1.4 times fewer obstacles than that of the NP and TP based approaches, respectively. However, the TP+OP approach and the OP approach retrieve a similar number of obstacles and trajectories.

Since the TP approach only considers trajectory pruning, the number of retrieved obstacles is the same as the baseline approach, NP, and the number of trajectories retrieved is nearly same as that of the TP+OP. On the other hand, the number of obstacles retrieved in the OP approach is nearly the same as that of the TP+OP approach, and the number of trajectories retrieved by OP is same as that of NP.

Los Angeles (LA): The effect of *k* for the LA dataset is shown in Fig. 12. As *k* increases, similar to the NY dataset, the performance of all of approaches almost remain constant. We observe that the TP+OP approach is about 20.6 times faster than NP, 18.2 times faster than TP, and 2 times faster than OP. Also the number of obstacles retrieved in TP+OP is about 3.7, 3.6, 1.7 times less than that of NP, TP, and OP approaches, respectively. The number of trajectories retrieved in TP+OP is about 4.7 times fewer than that NP and OP, and 2.2 times fewer than that of TP.

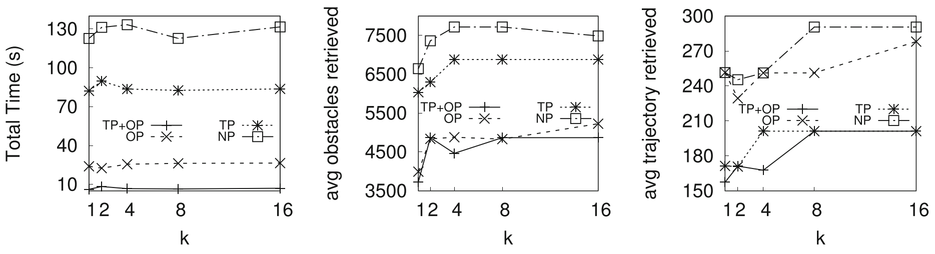

Fig. 11 Effect of *k* in New York dataset

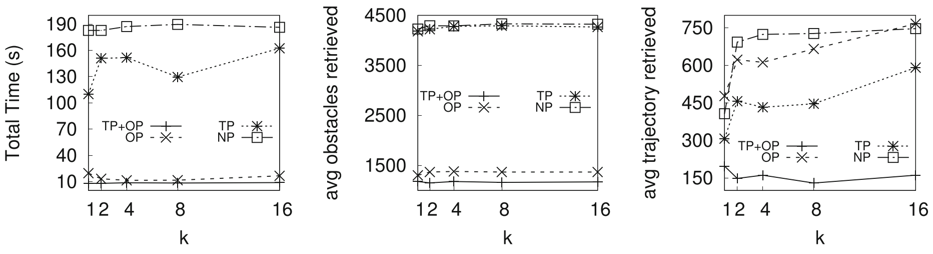

Fig. 12 Effect of *k* in Los Angles dataset

Boston (BN): Figure 13 shows the effect of k for the Boston downtown dataset, which shows similar performance as shown in two other datasets. The performance of the TP+OP approach is about 16, 11.5, 1.5 times faster than that of the NP, TP, and OP approaches, respectively. Also, The TP+OP approach retrieves about 2.5, 2, and 1.2 times fewer obstacles than that of the NP, TP, and OP approaches, respectively. We have also observed that the number of trajectories retrieved in the TP+OP approach is about 3, 2.8, and 1.7 times fewer than that of the NP, OP, and TP, respectively.

6.3.2 Effect of n_T

In this set of experiments, we vary the number of targets, n_T as 5, 10, 15, 20, and 25, and compare the performance of our methods in terms of processing time, and obstacle / trajectory retrieval costs for different datasets.

New York (NY): Figure 14 shows the effect of n_T for the NY dataset. As the number of targets increases, the total processing time increases for all methods. However, the graph is much steeper for all other methods when compared to the TP+OP approach. Here, the TP+OP approach is on average about 10, 7, 4.7 times faster than NP, TP, and OP approaches, respectively.

In general, we have observed increasing trends of the number number of obstacles and trajectories retrieved as n_T increases, which is expected. However, the number of obstacles retrieved in the TP+OP approach is about 2, and 1.7 times less than that of the NP and TP approaches, respectively. We have also observed similar trends for trajectory retrieval.

Los Angeles (LA): The effect of n_T for the LA dataset is shown in Fig. 15. As the number of targets increases, the total processing time increases for all methods. However, the

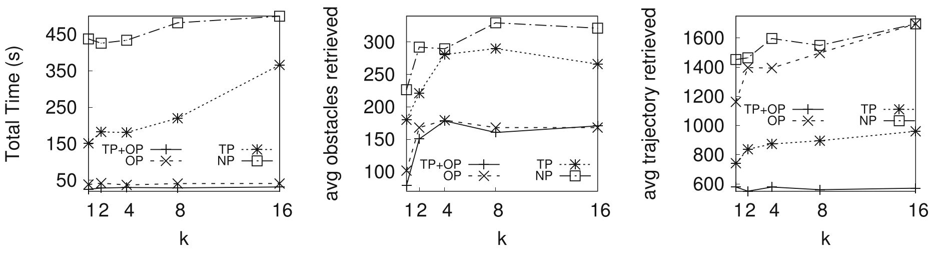

Fig. 13 Effect of *k* in (BN dataset)

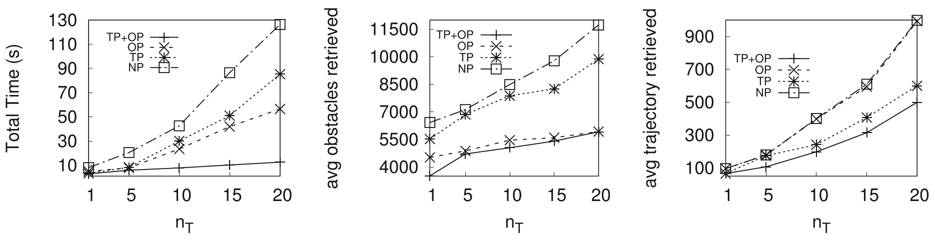

Fig. 14 Effect of n_T (NY dataset)

graph suggests exponential growth for the NP and TP approaches. Here, in term of processing time, the TP+OP approach is on average about 18 times faster than NP, 16 times faster than TP, and about 2.6 times faster than OP.

The number of obstacles and trajectories retrieved are increasing for all algorithms. For TP+OP, the number of obstacles retrieved is about 2.9 times less than NP, 2.4 times less than TP, and 1.1 times less than OP. The number of trajectories retrieved for TP+OP is about 1.6 times less than NP, 1.5 times less than OP, and 1.2 times less than TP.

Boston (BN): Figure 16 shows the effect of n_T for the BN dataset. As the number of targets increases, performance trends are similar to the NY and LA datasets. Here, TP+OP is on average about 40 times faster than NP, 30 times faster than TP, and about 2.2 times faster than OP. Moreover, the TP+OP approach retrieves about 3.9 times fewer obstacles than NP, and the OP and TP retrieve about 1.2 times and 3.1 times more obstacles, respectively than the TP+OP approach. The TP+OP approach retrieves about 3.2, 12.8, 1.2 times fewer trajectories when compared to the NP, OP, and TP approaches, respectively.

6.3.3 Effect of n_L

We have varied the number of trajectories n_L, and compared the performance of different methods. Since we have a fixed set of real trajectories for both NY and LA datasets, we only generate synthetic trajectories of various sizes, 0.5M, 1M, 2M, and 3M using the Boston and synthetic dataset.

Boston (BN): Figure 17 shows the results when varying n_L for the BN dataset. As n_L increases, the total processing time and number of trajectories retrieved increase for all methods. However, the graph is much steeper for NP and TP when compared to the

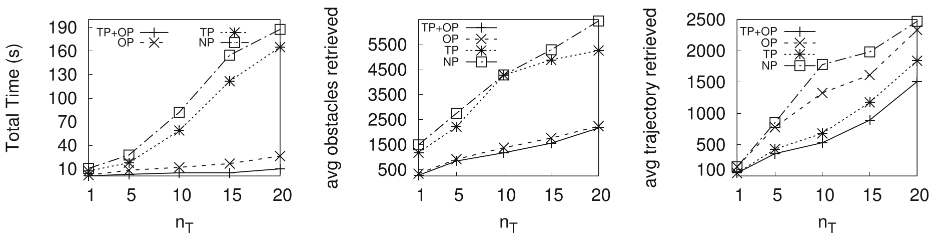

Fig. 15 Effect of n_T (LA dataset)

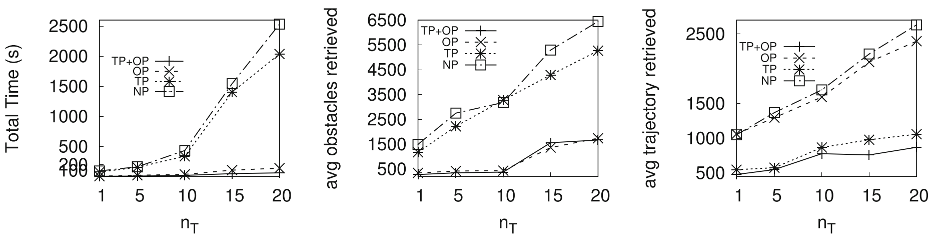

Fig. 16 Effect of n_T (BN dataset)

TP+OP and OP. Here, in term of processing time, the TP+OP approach is on average about 17 times faster than NP, 14 times faster than TP, and 2 times faster than OP.

Since with the increase of the number of trajectories, the *Trajectory Segmentation Range* of a target may also increase, resulting in more obstacles being retrieved. The number of obstacles retrieved is nearly identical for both TP+OP and OP, as both approaches used obstacle pruning. However, the TP+OP approach retrieves about 4 and 2.8 times fewer obstacles than that of the NP and TP approaches, respectively. The average number of trajectories retrieved is similar for both NP and OP, as neither of them prune trajectories. However, the TP+OP approach retrieves about 2.4 times fewer trajectories than both of these approaches.

Synthetic: The effect of n_O on the synthetic dataset is shown in Fig. 18. As the number of trajectories increases, the total processing time increases for all of the methods, and the trends ares similar to those shown in the Boston dataset. However, the TP+OP approach is on average about 56, 38, 2.2 times faster than the NP, TP, and the OP approaches, respectively. We have also observed that the average number of obstacles retrieved is nearly constant for both TP+OP and OP approaches and increases slightly for TP and NP approaches. We have also observed that the TP+OP approach always retrieve much fewer obstacles and trajectories than that of other approaches.

6.3.4 Effect of n_O

In this set of experiments, we vary the number of obstacles, n_O, as 0.5M, 1M, 2M, and 3M. Since real datasets have a fixed number of obstacles, we have generated a varying number obstacles synthetically and ran the experiments. Figure 19 shows that with the increase of

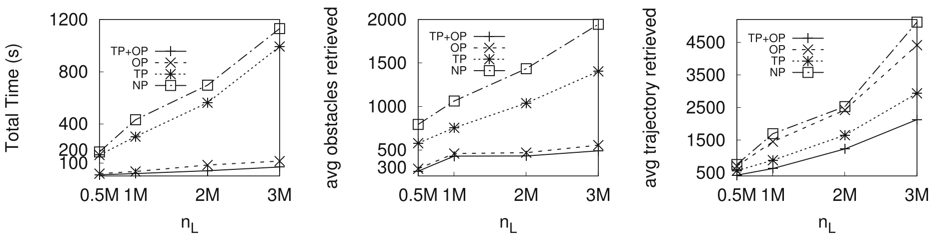

Fig. 17 Effect of n_L in Boston downtown dataset

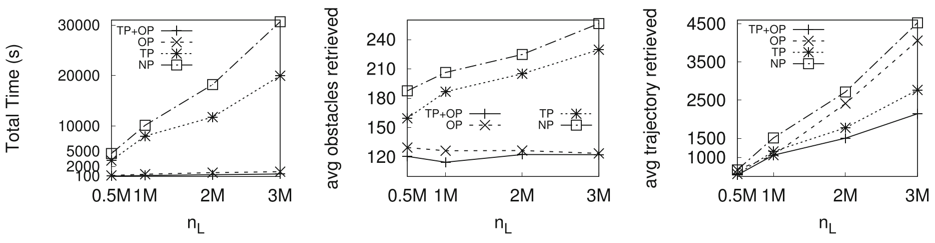

Fig. 18 Effect of n_L in Synthetic dataset

n_O, the total processing time and number of obstacles and trajectories retrieved, increase for all methods. This is expected as more obstacles mean more visibility computations in the query evaluation process. We also observe that the TP+OP approach is about 40, 29.5, 1.8 times faster than NP, TP, and OP approaches, respectively. Also, the TP+OP approach needs to retrieve much fewer obstacles and trajectories than all other methods.

6.3.5 Effect of A_T

In this set of experiments, we show the effect of varying the query (or target) area in Fig. 20. As A_T increases, the processing time increases for both the TP and NP approaches, and remains nearly same for both the OP and TP+OP approaches. However, the TP+OP approach is on average about 88, 77, and 1.8 times faster than NP, TP, and OP approaches, respectively.

The number of retrieved obstacles and trajectories also increase because if targets are concentrated in a smaller area, and they share common trajectories and obstacles in their visible range. Thus increasing A_T also increases the number of retrieved trajectories and obstacles. In all cases our TP+OP approaches outperform other approaches significantly.

6.3.6 Analysis

We have compared our three approaches: OP, TP, and OP+TP, with a baseline approach (NP). For our experimental results, it is evident that the TP+OP approach provides the best result in terms of processing time, number of obstacles and trajectories retrieved in all cases. We also observe that the OP approach outperforms the TP approach significantly in terms of processing time. The TP approach only employs trajectory pruning but no obstacle pruning. If we do not prune obstacles, then we have to take projection with respect to all obstacles

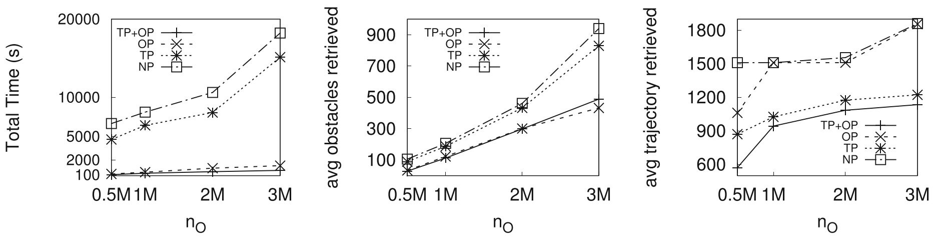

Fig. 19 Effect of n_O in Synthetic dataset

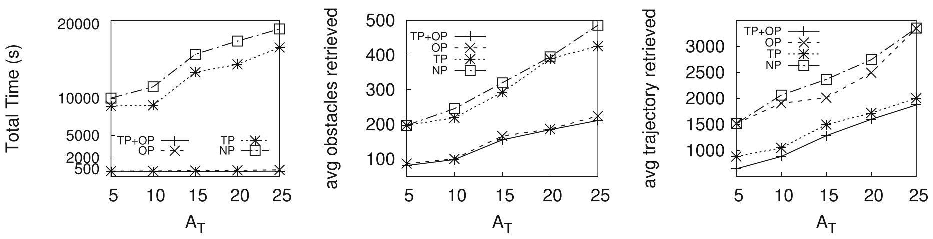

Fig. 20 Effect of A_T in Synthetic dataset

in the visible range which is computationally expensive. Since the OP approach employs obstacle pruning, but not trajectory pruning, we have to take projection with respect to the obstacles in the reduced obstacle set, which reduces the cost of taking projection every time.

In case of obstacle retrieval, the OP approach performs nearly the same as the TP+OP approach. In our approach, obstacles are pruned for each target independently while processing each *k*MVT query. Thus, no obstacles are pruned globally, and thereby the average number of obstacle retrieved is nearly same for them. However, in case of trajectory retrieval, the TP approach retrieves more trajectories than that of the TP+OP. The reason is as follows. Though both approaches consider trajectory pruning, the TP+OP approach also considers the target ordering and can therefore prune more trajectories as evaluating targets in a specific order can prune trajectories earlier in the process.

The performance of the OP approach in terms of the trajectory retrieval is nearly the same as that of the NP approach. If we do not consider trajectory pruning, then we have to consider all the trajectories inside the visible range. So nearly the same number of trajectories need to be retrieved in both approaches. But in case of obstacle retrieval, the TP approach performs better than the NP approach in all cases although none of them are using obstacle pruning. Since we consider trajectory pruning in TP, this approach provides a smaller *Trajectory Segmentation Range* for each target than that of the NP approach. Therefore the number of obstacles retrieved in the TP approach is less than that of the NP approach.

7 Conclusion

In this work, we have proposed and investigated the *k* Aggregate Maximum Visibility Trajectory (*k*AMVT) and its variants. The *k*AMVT query finds the trajectory that provides the best view of the targets in the presence of obstacles, which has many interesting applications that include VR guided navigation, tour planning, and advertisement. To efficiently solve the *k*AMVT query, we have proposed a series of optimization techniques such as obstacle and trajectory pruning mechanisms, and target ordering technique. To verify the efficiency and effectiveness of our solutions, we conduct an extensive experimental study using a number of large real and synthetic datasets: New York, Los Angles, Boston, and synthetic. Our experimental results show that our approach is between 10 and 50 times faster than the baseline approach. To the best of our knowledge, this is the first query-based solution that integrates user route activities or trajectories with the large-scale 3D modeling to answer an interesting set of visibility queries.

References

1. Asano T, Asano T, Guibas L, Hershberger J, Imai H (1985) Visibility-polygon search and euclidean shortest paths. In: Proceedings of the 26th annual symposium on foundations of computer science, SFCS. IEEE Computer Society, Washington, pp 155–164
2. Asano T, Asano T, Guibas L, Hershberger J, Imai H (1986) Visibility of disjoint polygons. Algorithmica 1(1):49–63
3. Ben-Moshe B, Hall-Holt O, Katz MJ, Mitchell JSB (2004) Computing the visibility graph of points within a polygon. In: Proceedings of the twentieth annual symposium on computational geometry, SCG '04. ACM, New York, pp 27–35
4. Bittner J (2002) Efficient construction of visibility maps using approximate occlusion sweep. In: Proceedings of the 18th spring conference on computer graphics, SCCG '02. ACM, New York, pp 167–175
5. Chen L, Ȯzsu MT, Oria V (2005) Robust and fast similarity search for moving object trajectories. In: Proceedings of the ACM SIGMOD international conference on management of data. Baltimore, Maryland, USA, June 14-16, pp 491–502
6. Chen Z, Shen HT, Zhou X (2011) Discovering popular routes from trajectories. In: Proceedings of the 27th international conference on data engineering, ICDE 2011, April 11-16. Hannover, Germany, pp 900–911
7. Chen Z, Shen HT, Zhou X, Zheng Y, Xie X (2010) Searching trajectories by locations: an efficiency study. In: Proceedings of ACM SIGMOD international conference on management of data, SIGMOD '10. ACM, New York, pp 255–266
8. Choudhury FM, Ali ME, Masud S, Nath S, Rabban IE (2014) Scalable visibility color map construction in spatial databases. Inf Syst 42:89–106
9. Ding X, Chen L, Gao Y, Jensen CS, Bao H (2018) Ultraman: a unified platform for big trajectory data management and analytics. Proceedings of the VLDB Endowment 11(7):787–799
10. Erikson C, Manocha D, Baxter WV III (2001) Hlods for faster display of large static and dynamic environments. In: Proceedings of symposium on interactive 3D graphics. ACM, pp 111–120
11. Gao Y, Zheng B (2009) Continuous obstructed nearest neighbor queries in spatial databases. In: SIGMOD Conference. ACM, pp 577–590
12. Gao Y, Zheng B, Chen G, Li Q, Chen C, Chen G (2010) Efficient mutual nearest neighbor query processing for moving object trajectories. Inform Sci 180(11):2176–2195
13. Gao Y, Zheng B, Chen G, Li Q, Guo X (2011) Continuous visible nearest neighbor query processing in spatial databases. VLDB J 20(3):371–396
14. Gao Y, Zheng B, Lee WC, Chen G (2009) Continuous visible nearest neighbor queries. In: Proceedings of the 12th international conference on extending database technology: advances in database technology, EDBT '09. ACM, New York, pp 144–155
15. Gu Y, Yu X, Yu G (2014) Method for continuous reverse k-nearest neighbor queries in obstructed spatial databases 25:1806–1816
16. Guttman A (1984) R-trees: a dynamic index structure for spatial searching, vol 14. ACM
17. Haider CMR, Arman A, Ali ME, Choudhury FM (2016) Continuous maximum visibility query for a moving target. In: Australasian database conference. Springer, pp 82–94
18. Heffernan PJ, Mitchell JSB (1995) An optimal algorithm for computing visibility in the plane. SIAM J Comput 24(1):184–201
19. Kalashnikov DV, Prabhakar S, Hambrusch SE (2004) Main memory evaluation of monitoring queries over moving objects. Distrib Parallel Datab 15(2):117–135
20. Kim DS, Yoo KH, Chwa KY, Shin SY (1998) Efficient algorithms for computing a complete visibility region in three-dimensional space. Algorithmica 20(2):201–225
21. Lee KC, Lee WC, Zheng B (2009) Fast object search on road networks. In: Proceedings of the 12th international conference on extending database technology: advances in database technology. ACM, pp 1018–1029
22. Levandoski JJ, Khalefa ME, Mokbel MF (2011) The caredb context and preference-aware database system. In: 5th International workshop on personalized access, profile management, and context awareness in databases, PersDB-in conjunction with very large data bases, VLDB
23. Levandoski JJ, Mokbel MF, Khalefa ME (2010) Flexpref: a framework for extensible preference evaluation in database systems. In: IEEE 26th International conference on data engineering (ICDE). IEEE, pp 828–839
24. Masud S, Choudhury FM, Ali ME, Nutanong S (2013) Maximum visibility queries in spatial databases. In: 29th International conference on data engineering (ICDE). IEEE, pp 637–648

25. Mouratidis K, Lin Y, Yiu ML (2010) Preference queries in large multi-cost transportation networks. In: IEEE 26th International conference on data engineering (ICDE). IEEE, pp 533–544
26. Mouratidis K, Papadias D, Hadjieleftheriou M (2005) Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In: Proceedings of ACM SIGMOD international conference on management of data. ACM, pp 634–645
27. Nutanong S, Tanin E, Zhang R (2007) Visible nearest neighbor queries. Springer, Berlin, pp 876–883
28. Nutanong S, Tanin E, Zhang R (2010) Incremental evaluation of visible nearest neighbor queries. IEEE Trans Knowl Data Eng 22(5):665–681
29. Papadias D, Zhang J, Mamoulis N, Tao Y (2003) Query processing in spatial network databases. In: Proceedings of the 29th international conference on very large data bases, vol 29. VLDB Endowment, pp 802–813
30. Rabban IE, Abdullah K, Ali ME, Cheema MA (2015) Visibility color map for a fixed or moving target in spatial databases. In: International symposium on spatial and temporal databases. Springer, pp 197–215
31. Rocha-Junior JB, Nørvåg K (2012) Top-k spatial keyword queries on road networks. In: Proceedings of the 15th international conference on extending database technology. ACM, pp 168–179
32. Shafique S, Ali ME (2016) Recommending most popular travel path within a region of interest from historical trajectory data. In: Proceedings of the 5th ACM SIGSPATIAL international workshop on mobile geographic information systems. ACM, pp 2–11
33. Shang S, Ding R, Yuan B, Xie K, Zheng K, Kalnis P (2012) User oriented trajectory search for trip recommendation. In: Proceedings of the 15th international conference on extending database technology, EDBT. ACM, New York, pp 156–167
34. Shang S, Ding R, Zheng K, Jensen CS, Kalnis P, Zhou X (2014) Personalized trajectory matching in spatial networks. VLDB J 23(3):449–468
35. Shou L, Huang Z, Tan KL (2003) Hdov-tree: the structure, the storage, the speed. In: Proceedings on 19th international conference on data engineering. IEEE, pp 557–568
36. Song Z, Roussopoulos N (2001) K-nearest neighbor search for moving query point. In: International symposium on spatial and temporal databases. Springer, pp 79–96
37. Stewart AJ, Karkanis T (1998) Computing the approximate visibility map, with applications to form factors and discontinuity meshing. Springer, Vienna, pp 57–68
38. Suri S, O'Rourke J (1986) Worst-case optimal algorithms for constructing visibility polygons with holes. In: Proceedings of the second annual symposium on computational geometry, SCG '86. ACM, New York, pp 14–23
39. Tao Y, Papadias D (2002) Time-parameterized queries in spatio-temporal databases. In: Proceedings of the 2002 ACM SIGMOD international conference on management of data. ACM, pp 334–345
40. Tao Y, Papadias D, Shen Q (2002) Continuous nearest neighbor search. In: VLDB'02: Proceedings of the 28th international conference on very large databases. Elsevier, pp 287–298
41. Tao Y, Papadias D, Shen Q (2002) Continuous nearest neighbor search. In: Proceedings of the 28th international conference on very large data bases, VLDB '02. VLDB Endowment, pp 287–298
42. Tsai YHR, Cheng LT, Osher S, Burchard P, Sapiro G (2004) Visibility and its dynamics in a pde based implicit framework. J Comput Phys 199(1):260–290
43. Wang S, Bao Z, Culpepper JS, Sellis T, Cong G (2017) Reverse k nearest neighbor search over trajectories. IEEE Transactions on Knowledge and Data Engineering
44. Xia C, Hsu D, Tung AKH (2004) A fast filter for obstructed nearest neighbor queries. Springer, Berlin, pp 203–215
45. Yu X, Pu KQ, Koudas N (2005) Monitoring k-nearest neighbor queries over moving objects. In: Proceedings on 21st international conference on data engineering, iCDE. IEEE, pp 631–642
46. Zarei A, Ghodsi M (2005) Efficient computation of query point visibility in polygons with holes. In: Proceedings of the twenty-first annual symposium on computational geometry, SCG '05. ACM, New York, pp 314–320
47. Zhang C, Shou L, Chen K, Chen G (2012) See-to-retrieve: efficient processing of spatio-visual keyword queries. In: SIGIR, pp 681–690
48. Zhang D, Ding M, Yang D, Liu Y, Fan J, Shen HT (2018) Trajectory simplification: an experimental study and quality analysis. Proc VLDB Endowment 11(9):934–946
49. Zhang J, Papadias D, Mouratidis K, Zhu M (2004) Spatial queries in the presence of obstacles. Springer, Berlin, pp 366–384
50. Zheng K, Shang S, Yuan NJ, Yang Y (2013) Towards efficient search for activity trajectories. In: ICDE. IEEE Computer Society, pp 230–241

Nafis Irtiza Tripto is currently a Lecturer in the Department of Computer Science and Engineering (CSE) at Bangladesh University of Engineering and Technology (BUET). He has completed his B.Sc. degree from same department in September, 2017 and currently enrolled as a Masters student.

His research areas particularly focus on Spatial data, Machine Learning and Textual data analysis.

Mahjabin Nahar is currently a Lecturer in the Department of Computer Science and Engineering (CSE) at Bangladesh University of Engineering and Technology (BUET). Her research interests focus on Artificial Intelligence, Machine Learning, Spatial Databases.

Dr. Mohammed Eunus Ali is a Professor in the Department of Computer Science and Engineering (CSE) at Bangladesh University of Engineering and Technology (BUET), Dhaka since May 2014. He received his PhD degree in Computer Science and Software Engineering from the University of Melbourne in 2010. He also worked as a Research Fellow and Visiting Research Scholar in the University of Melbourne in 2010 and 2012-2013, respectively. He was also visiting research fellow at Monash University and RMIT University in 2015 and 2016, respectively. Dr. Eunus is the recipient of prestigious UGC Award in the year 2012 for his outstanding research contribution.

Dr. Eunus's research falls in the intersection of data management and mobile computing. His research areas cover a wide range of topic in database systems and information management that include spatial and multimedia databases, location based services, social media and big data analytics. The primary focus of his current research is developing systems and algorithms to improve the performance of location based services in mobile environments.

Farhana Murtaza Choudhury is a Lecturer (Early Career Development Fellow) at RMIT University in the School of Science (Computer Science and Information Technology) since September, 2017. She is affiliated with the RMIT - NICTA Data Analytics Lab and the Information Storage and Retrieval (ISAR) Research Group ran by Professor Mark Sanderson.

Her current work focuses on Spatial and Spatial-textual Databases, Big Data, Social network, and Streaming query processing. Her PhD supervisors are Prof. Timos Sellis and Dr. Shane Culpepper.

Dr. J. Shane Culpepper is a Vice-Chancellor's Principal Research Fellow and Associate Professor (Reader in the UK, Full Professor in North America) at RMIT University in the School of Science (Computer Science). He runs the Information Discovery Lab, and is a member of the Information Storage and Retrieval (ISAR) Research Group.

His current work focuses on designing and evaluating efficient and effective algorithms and data structures for a wide variety of information storage and retrieval problems. Broadly speaking his research interests include information retrieval, text indexing, data compression, system evaluation, information discovery, learning to rank, natural language processing, algorithm engineering, and scalable distributed/parallel computing.

Professor Timos Sellis is Director of Swinburne's Data Science Research Institute. His research interests include big data, data streams, personalisation, data integration, and spatio- temporal database systems.

Professor Sellis is an IEEE Fellow for his contributions to database query optimisation and spatial data management. He is also an ACM Fellow for his contributions to database query optimisation, spatial data management and data warehousing.

Up until 2012 Professor Sellis was Director of the Institute for the Management of Information Systems and a Professor at the National Technical University of Athens. He holds an MSc degree from Harvard University, a PhD from the University of California at Berkeley, and has served as president of the National Council for Research and Technology of Greece.