# Multi-location Visibility Processing for a Moving Target

Komallapalli Kaushik [*]
IIIT Hyderabad
Hyderabad, India
kaushik.k@research.iiit.ac.in

P. Krishna Reddy [†]
IIIT Hyderabad
Hyderabad, India
pkreddy@iiit.ac.in

Anirban Mondal [‡]
Ashoka University
Delhi, India
anirban.mondal@ashoka.edu.in

## ABSTRACT

Visibility computation of moving targets is one of the critical areas of research due to routine change in motion of target. The goal of visibility computation of a moving target is to efficiently maximise the visibility of target at every instant of time. Incidentally, research efforts are being made to address the visibility computation of moving targets where it identifies individual locations for maximizing the visibility of a given target object. However, in many applications, a set of locations may be more effective than just individual locations towards maximizing the visibility of the given target object. Existing work addresses the problem of identifying multi locations for maximizing the visibility of a static target object. In this paper, we introduce the Continuous Multi-Location Visibility query ($CMLV$). An $CMLV$ query determines the top-k query locations from which the visibility of a moving target object can be maximized. We propose a pre-computed transactional based framework and incremental coverage pattern mining based algorithm to process $CMLV$ queries. The results of our performance evaluation with real datasets demonstrates that our proposed approach $CMLV$ query indeed improves the performance significantly w.r.t. processing $MLV$ query for every time instant.

**PVLDB Reference Format:**
Komallapalli Kaushik, P. Krishna Reddy and Anirban Mondal. Multi-location Visibility Processing for a Moving Target. *PVLDB*, 12(xxx): xxxx-yyyy, 2020.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

## 1. INTRODUCTION

Visibility computation has attracted great research interests due to emerging applications in spatial databases, computer graphics, computational geometry and geographic information systems. Visibility computation involves finding the visibility of the target from a query location. In many applications the target to be monitored is not stationary but continuously moving; this necessitates fast and scalable algorithms. In the literature algorithms have been proposed for identifying the top-$k$ individual query locations that maximize the visibility of the moving target object [14] and so on. In this paper we determine top-$k$ candidate sets of query locations from which the visibility of a moving target object can be maximized in the presence of obstacles.

Visibility computation has important applications in the area of security, good viewpoints selection. Consider a defence-related surveillance application scenario. Here, the visibility of the target object, which needs to be kept under observation, typically changes depending upon the environment (e.g., obstacles such as trees and the nature of the terrain) and the movement of the target object. More concrete examples can be given as finding suitable locations to enjoy maximum visibility of the first car in car racing whereas the car keeps on moving the visibility changes w.r.t to every location. visibility computation is essential for surveillance applications such as monitoring car parking lots by means of CCTV cameras. Observe that all of these above application scenarios involve the presence of obstacles such as buildings, trees and so on.

Existing works [8, 10, 24, 17] have addressed the problem of visibility computation in spatial databases in the presence of obstacles continuously to determine k-nearest neighbour ($CkNN$) queries. However, these works do not address visibility maximization of a moving target object. In contrast, given a set of candidate locations, the Continuous Maximum Visibility ($kCMV$) query [14] determines a ranked list of the top-$k$ locations from which the visibility of a moving target object can be maximized in the presence of obstacles. However, existing approaches, such as [14], suffer from a drawback in that they try to maximize the visibility of a moving target object from a single query location. However, in real-world scenarios, it often becomes practically infeasible to cover a significant percentage of any moving target object from only a single query location, thereby narrowing the applicability of the existing approaches. Intuitively, a combination of query locations could provide significantly higher visibility of a moving target as opposed to any one single query location. An effort has been made in [6] ($MLV$) by considering the combination of query locations for maximizing the visibility of a given target. In this paper, we have extended the approach proposed in [6] by considering the combination of query locations for maximizing the visibility of a moving target. However, given $n$ query lo-

[*]First author
[†]Second author
[‡]Third author


*Proceedings of the VLDB Endowment,* Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

cations and moving target the main issue is to update the *MLV* query result at every time instant as the target moves, which incurs significant computation costs.

In this paper, we propose a *CMLV* query for determining top-$k$ candidate sets of query locations from which the visibility of a moving target object can be maximized in the presence of obstacles. We assume that the query locations are points, while the obstacles and the given target object are regions in space. The main issue is to update the *MLV* query result at every time instant as the target moves, which incurs significant computation costs. To overcome this main problem of high computation head for updating *MLV* query for every timestamp we proposed a technique in which precompute the portion transactions of the target and incremental coverage pattern mining based algorithm to process *CMLV* queries. To our knowledge this is the first work to determine candidate sets of query locations from which the visibility of a moving target object can be maximized in the presence of obstacles.

To determine a set of multiple query locations to maximize the visibility of a moving target object $T$, we introduce the Continuous Multi-Location Visibility (*CMLV*) query. Given $T$, a set of query locations, a set of obstacles and the path in which the target is moving the *CMLV* query determines the top-$k$ candidate sets of query locations for every timestamp. Each candidate set comprises possibly multiple query locations, which together maximize the visibility of $T$, while satisfying the threshold values of visibility and overlap and ensuring continuity. From these top-$k$ candidate sets, users can choose one or more appropriate candidate sets depending upon the application requirements.

We divide the whole path in which target $T$ is moving in equal-sized units,which we designate as portions. The size of a portion is essentially application-dependent. By dividing the target path into portions, the target path is represented as a set of portion identifiers (*pids*). We assume that the size of target is less than the size of path in which the target is moving. While the target $T$ is moving in the path it also covers the few number portions in which the target is divided. Thus, now the visibility of $T$ is based on the portions that it covers in the path in which it is moving as opposed to the area of the regions of $T$, thereby replacing complex computationally-intensive spatial operations associated with visibility computations by simpler set-based operations.

Now form portion transactions of the target at every time instant by modelling the association of the respective portion that is overlapped in the path portions with the corresponding query locations from which the portion is visible as a transaction. Thus, the portion transactions represent all of the possible associations of portions and query locations. For efficiently extracting the combinations of query locations from portion transactions at every time instant, we propose an algorithm based on incremental mining of coverage patterns.

The key contributions of this work are three-fold:

1. We introduce continuous multi location visibility query (*CMLV*).

2. We introduce the pre-computation technique to reduce the number of query locations and obstacles to be considered for the current position of the target.

3. We present an efficient algorithm for processing CMLV queries that reduces the computation cost. The algorithm incrementally outputs the multi locations for every time instant.

## 2. RELATED WORK AND BACKGROUND

Visibility computation approaches in spatial databases [8, 9, 10, 20, 24, 25] apply the concept of $k$ nearest neighbours ($k$NN). A $k$NN query finds the $k$ nearest points from a given query point based on a distance measure. The work in [8] addressed continuous obstructed nearest neighbor queries by using the concept of control points in conjunction with an efficient quadratic-based split point computation algorithm for processing only the relevant data points and obstacles. The visible reverse nearest neighbor (VRNN) query [9] considers the impact of obstacles on the visibility of objects by pruning away irrelevant data objects to save the traversal cost. Given a query point $q$, a VRNN query retrieves the points that have $q$ as their visible nearest neighbor in the presence of obstacles. The work in [20] incrementally computes visible neighbors, while enlarging the search space. The work in [7] examined the Obstructed Reverse Nearest Neighbor query, which is a variant of the reverse nearest neighbour query that also considers the presence of obstacles.

Continuous nearest neighbor (CNN) queries [24] involve $k$NN query processing for a moving query point. CNN algorithms use branch-and-bound techniques for pruning. The work in [10] proposed variants of the continuous visible nearest neighbor search by indexing both data points and obstacles in R-trees separately. The work in [15] introduced the $k$ Continuous Maximum Visibility query, which determines the top-$k$ query locations (from a set of candidate query locations) sorted in the order of visibility of a given target from these query locations. The Group Visible Nearest Neighbor query in [25] considered both visibility and distance as constraints.

Visibility problems concerning terrains have been discussed in [5]. Visibility computations on a terrain may involve either one or multiple viewpoints and range from visibility queries to the computation of structures that encode the visible portions of the surface. In [12], visibility coverage problems were defined by using visibility information derived from topographic surfaces. Furthermore, the HDoV-tree index [22] has been proposed to improve visual fidelity and performance in large virtual environments based on the degree of object visibility. The proposal in [2] indicated how to create a data structure for computing the visibility polygon of a given point w.r.t. the set of polygons.

The concept of coverage has been used to solve node cover problem in graph theory [11] and set cover problem in set theory [4]. The model of coverage patterns and a level-wise pruning approach to extract coverage patterns from transactional databases was proposed in [23]. Pattern growth approach to extract coverage patterns was proposed in [13].

Given a target object $T$ and a set of query locations, the work in [18] proposed the MV query. The MV query determines a ranked list of *individual query locations* that maximize the visibility of $T$ in the presence of obstacles. The work in [18] proposed some variants for processing MV queries. Among these variants, the Query centric Distance based variant (*QD*) outperformed the others. In *QD*, the obstacles are incrementally considered according to their non-

decreasing distances from the query location currently under consideration. In this way, a query point that is severely blocked by a subset of obstacles can effectively be ruled out from the search process.

Approaches for maximizing visibility also include the works in [1, 16]. The work in [16] proposed the $k$ Aggregate Maximum Visibility Trajectory query, which determines the top-$k$ trajectories that provide the best view of the target objects in the presence of obstacles. The work in [1] examines the Maximum Visibility Facility Selection query, which selects $k$ locations (from a candidate set of locations) to place a facility (e.g., a surveillance camera) using which the visibility of a given region can be maximized.

Notably, *none* of these existing approaches has addressed the issue of determining a set of *multiple* query locations for maximizing the visibility of a given target object. Moreover, our work differs from that of the work in [18], which only uses *individual* query locations for maximizing the visibility of a given target object. Additionally, we have proposed an efficient approach for processing MLV queries by means of a portion-based framework, forming portion transactions and extending the concept of coverage patterns.

## 2.1 Background information about visibility

Similar to the visibility computation approach in [18], we adopt the definition of visibility from the works in [19, 21]. Given two point locations $l_i$, $l_j$ and a set $O$ of obstacles in space, $l_i$ and $l_j$ are considered to be visible to each other if the straight line connecting them does not intersect with any of the obstacles in $O$. Let $Q$ be the set of all point query locations i.e., $Q = \{q_1, q_2, q_3, \dots q_n\}$, where $n$ is the total number of query locations. A region $V$ is considered to be visible w.r.t. a given query location $q_i$ iff every point $v$ in $V$ has point-to-point visibility w.r.t. $q_i$. The visibility of an object $T$ from query location $q_i$ (designated as $vis(q_i, T)$) is defined as the ratio of the angular size $AS(T)$ of $T$ to the original size $OS(T)$ of $T$.

$$vis(q_i, T) = \begin{cases} AS(T)/OS(T), & \text{if } d <= maxD \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

In Equation 1, observe that for $T$ to be visible from $q_i$, the minimum distance between $T$ and $q_i$ should be below a certain threshold, which we designate as $maxD$. Here, $maxD$ is essentially application-dependent. For example, if a human is viewing $T$ from $q_i$ with his naked eye, the value of $maxD$ could be set to at most 1 km. On the other hand, the value of $maxD$ could be set in the range of a few km when the human is viewing $T$ from $q_i$ with binoculars.

Now let us define the notion of minimum visibility threshold $minV$. The value of $minV$ is specified by the users depending upon their requirements. Based on the $minV$ threshold, all individual query locations, for which $vis(q_i, T) \geq minV$, will be part of the result.

We denote the set of the given region(s) of $T$ visible from $q_i$ as $VR(q_i, T)$. (Multiple regions of T may be visible from a given $q_i$). We use Area() function to represent the spatial area covered by a spatial object or a set of spatial regions. Consider a set $\psi$ of $n$ query locations, where $q_i$ represents the $i^th$ query location. We define $vis(\psi, T)$ (i.e., the visibility of T w.r.t. $\psi$) as follows:

$$vis(\psi, T) = Area( \bigcup_{i=1}^{n} VR(q_i, T) \ ) \ / \ Area(T) \quad (2)$$

In Equation 2, the term $Area( \bigcup_{i=1}^{n} ( VR(q_i, T)) \ )$ represents the area of the union of the visible regions of $q_i$, where $i$=1 to $n$. Consequently, for $\psi$ to contribute to the $CMLV$ query result, $vis(\psi, T) \geq minV$. Regarding overlap, it is possible for the same region of $T$ to be visible from multiple query locations. This requires approaches for identifying combinations of query locations that would minimize overlap among the visible regions of $T$ w.r.t. Individual query locations. We designate the overlap for candidate set $c_i$ as $overlap(c_i, T)$. The value of $overlap(c_i, T)$ is computed below:

$$overlap(c_i, T) = Area\Big(\sum_{i=1}^{n_i} VR(q_i, T)\Big) - Area\Big(\bigcup_{i=1}^{n_i} VR(q_i, T)\Big) \quad (3)$$

In Equation 3 $Area(\sum_{i=1}^{n_i} VR(q_i, T))$ is the sum of the areas of visible regions of $q_i$ where $i = 1$ to $n_i$. For $c_i$ to be $CMLV$ query result, $overlap(c_i, T) \leq maxO$, where $maxO$ is the user-defined maximum overlap threshold.

Regarding *continuity*, given query locations $q_i$ and $q_j$ from the set $Q$ of $n$ query locations, $q_i$ and $q_j$ form one-to-one continuity relationship if there is a non-zero overlap between $VR(q_i, T)$ and $VR(q_j, T)$. We consider $q_i$ and $q_j$ to be continuous if there is a transitive-closure of one-to-one continuity relationship between $q_i$ and $q_j$ over $Q$. We designate the continuity for candidate set $c_i$ as $continuity(c_i, T)$. Thus, $continuity(c_i, T)$ is *true* iff the visible regions of every pair of query locations in $c_i$ are continuous; otherwise, it is $false$.

## 2.2 MV Query

In this paper, they introduced a novel form of spatial queries, called the Maximum Visibility ($MV$) query. The $MV$ query returns top-$k$ query locations in the order of visibility of the target. To efficiently process the query, they introduced three different approaches: query centric distance based approach ($QD$), query centric visible region based approach ($QV$) and target centric distance based approach ($TD$). Starting from a query point, the $QD$ and $QV$ approaches retrieve obstacles incrementally based on their distances from the query point and based on their effects on visible regions, respectively. On the contrary, $TD$ approach retrieves obstacles incrementally based on their distances from the target.

## 2.3 CMV Query

In this paper, they introduced a new type of query, namely $k$ Continuous Maximum Visibility Query that continuously finds the best viewpoint for a moving target in the presence of obstacles. To efficiently answer a $kCMV$ query, they introduced two approaches. The first approach, AVR-BS, relies on pre-computation to provide fast answers to queries. On the other hand, the second approach, VR-BS, makes a tradeoff between pre-computation and memory usage while processing queries.

## 2.4 MLV Query

In this paper, they introduced a new query called Multi-Location Visibility ($MLV$) query. The $MLV$ query returns the top-$k$ candidate sets in the order of visibility of the target where each candidate set contains a set of query locations. To efficiently process the query, they introduced a portion-based transactional framework and coverage pattern mining based algorithm. The approach includes division the target into a set of portions which reduces the problem into a set problem. Further each portion will be associated with a set of query points from which that particular portion is visible which are called as portion transactions. From portion transactions top-$k$ candidate sets are extracted by coverage pattern mining based algorithm.

# 3. CONTEXT OF THE PROBLEM

Consider a target object $T$ moving in the path $TP$, a set $Q$ of query points, a set $O$ of obstacles, a minimum visibility threshold $minV$, a maximum overlap threshold $maxO$ and $continuity = true$. The $CMLV$ query returns a set $C$ of top-$k$ candidate sets at every instant of time. Each candidate set $c_i$ (where $c_i \in C$) comprises a set of query locations, which maximize the visibility of $T$, while satisfying the $minV$, $maxO$ and $continuity$ criteria. Recall our discussions about the $minV$, $maxO$ and $continuity$ constraints in section 2.

Multiple candidate sets may qualify with the given $minV$, $maxO$ and $continuity$ constraints. We perform top-$k$ ranking of these candidate sets as follows. Since visibility maximization is our key focus, we primarily rank the candidate sets in descending order of their respective values of visibility. If more than one candidate set has the same value of visibility, priority is given to the candidate set with lower overlap. If multiple candidate sets have equal values for both visibility and overlap, ties are resolved based on the number of query locations in the set i.e., the one with lower number of query locations is ranked higher. Now we define the $CMLV$ query as follows:

**Continuous Multi Location Visibility (CMLV) query** Given a moving target object $T$ that is at location $l_i$ at time instance $t_i$ in a d-dimensional space $R^d$, a set $Q = \{q_1, q_2, q_3, \ldots q_n\}$ of $n$ query locations, a set $O$ of obstacles and the user-specified threshold values of $minV$, $maxOR$, $k$, the $CMLV$ query produces the following output. At each time instance $t_i$, the $CMLV$ query returns a set $C = \{c_1, c_2, c_3, \ldots c_k\}$ of top-$k$ candidate sets, where each candidate set $c_i$ comprises a set of query locations such that $vis(c_i, T) \geq minV$, $overlap(c_i, T) \leq maxOR$ and $continuity(c_i, T) = true$.

# 4. PROPOSED APPROACH

This section presents our proposed approach.

## 4.1 Basic idea

Our goal is to develop an approach for determining combinations of query locations for a moving target at every time instant. The main challenge is that since the target object is moving, we need to re-compute $MLV$ query and update the ranking of combination of query points for every position change of the target. A straightforward approach is to run the $MLV$ query every time when target $T$ moves. This approach results in high processing and computational overhead as for each position change of the target we need to consider all obstacles and query points for calculating portion transactions. Since there can be many common obstacles that fall within the range of two consecutive target locations, the retrieval of the same object occurs multiple times. To overcome the above limitation, we introduce an approach called $CMLV$ in which we rely on pre-computed portion transactions based on our target's path, obstacle and query datasets.

The basic idea of $CMLV$ is as follows. We divide a given target's path in which target $T$ moves into equal-sized units, which we designate as portions. Thus, we model target's path as a set of portions. Each portion has a unique identifier $pid$ and the size of these portions is application-dependent. Then we efficiently find the query locations and its respective visible portions for the whole target's path. Now we convert the query locations and its respective portions into portion transactions. We refer this step as a pre-computation of portion transactions.

Intuitively, the pre-computation step provides opportunities for efficient determination of top-k candidate sets as well as for efficient computations of visibility, overlap and continuity for each candidate set. The computation of the visible regions of target $T$ from a given query location becomes equivalent to computing the corresponding set of visible $pids$ of target's path that target $T$ is overlapping with. Furthermore, the visibility of $T$ from a combination of query locations can be computed as the union of the corresponding sets of the visible $pids$ of the target's path. Thus, we are essentially replacing complex and computationally expensive spatial operations by set-based operations, which are typically faster by several orders of magnitude. In a similar vein, computations of overlap and continuity are based on set-based operations, which are also faster by several orders of magnitude as compared to that of complex spatial operations.

So after knowing the location of the target $T$ by seeing overlapping portions of the path with target, we just form the portion transactions of the target at that time instant. Hence, the problem of extracting candidate sets becomes the problem of extracting set of query locations from the set of portion transactions of the target. To extract set of query locations from portion transactions at every time instant from scratch is highly computational. To overcome this limitation we propose an incremental pattern-based extraction method by exploiting an overlap-related pruning heuristic, which we shall discuss now.

The approach is, we consider the previous set of candidates to find the new set of candidates when the target is moved from one location $l_i$ to another $l_j$. When the target is moving from one location to another there might be many common portion transactions between them. So let us consider the set of transactions at location $l_i$ as DB and set of transactions at location $l_j$ as D'. so using the existing knowledge (i.e set of candidate at $l_i$), the portion transactions at $l_i$ and the portion transactions at $l_j$, we can compute the set of candidates at $l_j$ efficiently. The candidate set should satisfy the maximum overlap threshold, minimum visibility threshold and continuity constraints.

## 4.2 CMLV query processing

Now we shall discuss the proposed approach for $CMLV$ query processing as follows:

1. Division of target's path into portions.

2. Pre-computation of portion transactions of target's path.

3. Finding portion transactions of target $T$.

4. Determining the top-$k$ candidate sets incrementally.

### 4.2.1 Division of target's path into portions

We first explain the concept of portions. The target path ($TP$) is divided into a set $P$ of $n_p$ equal-sized units, which we designate as portions. Thus $p = p_1, p_2, p_3, \ldots, p_{n_p}$, where $p_i$ represents a given portion with unique identifier $pid_i$. For any pair of $p_i$ and $p_j$, $p_i \cap p_j = \varnothing$. Moreover, $p_1 \cup p_2 \cup \ldots \cup p_{np} = TP$.

For simplicity, the notion of a portion in 2D is as follows. As the visible area of $TP$ in 2D forms a line segment, a portion in 2D is part of a line segment. Thus, we denote $p_i$ as a pair of two points. Portion $p_i$ is represented as a line segment with the pair of two coordinates. So, $p_i = \langle((x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}))\rangle$, where $(x_{i_1}, y_{i_1})$ is the start point of $p_i$, represented by $sp(p_i)$ and $(x_{i_2}, y_{i_2})$ is the end-point of $p_i$, represented by $ep(p_i)$. So, $p_i = \langle sp(p_i), ep(p_i)\rangle$. Notably, the definition of a portion can also be extended to 3D albeit with some modifications.

Algorithm 1 depicts the steps of dividing the target path $TP$ into $w$ portions. We assume that target's path will be a set of line segments. We approximate $TP$ by its MBR. We divide the line segments of the target path $TP$ into portions. It may be possible that length of line segments of $TP$ may not be integral multiples of portion size $PS$. In such cases, the leftover part is considered as a single portion even though its length is less than $PS$. The *pids* are assigned to portions by adding 1 to the preceding *pid*, starting from 0.

**Algorithm 1:** Division of Target's path into portion

**Input :** Target Path $TP$, number of portions $n_p$
**Output:** Set $P$ of portions of target's path $TP$
line segment in $TP \leftarrow \langle sp(s), ep(s)\rangle$;
$sp(s), ep(s)$ are start and end-points of line segment;
Portion size $PS \leftarrow T/n_p$;
Initialize $d$ is the length of line segment;
Initialize $P$ to empty set, $k$ to 0, $d_t$ to $PS$;
Initialize $(cur.x, cur.y) to (sp(s).x, sp(s).y)$;
1: **for** each line segment $\langle sp(s), ep(s)\rangle$ in $TP$ **do**
2: **while** $k < n_p$ **do**
3: $t = d_t/d$
4: $temp.x = (sp(s).x + t * (ep(s).x - sp(s).x))$
5: $temp.y = (sp(s).y + t * (ep(s).y - sp(s).y))$
6: $P = P \cup \langle(cur.x, cur.y), (temp.x, temp.y)\rangle$
7: $(cur.x, cur.y) = (temp.x, temp.y)$
8: $d_t = d_t + PS$
9: $k = k + 1$
10: **end while**
11: **end for**
return $P$;

We discuss the visibility $vis(q_i, p_j)$ of a portion $p_j$ from a query location $q_i$ and the visible region $VR(q_i, TP)$ of $q_i$. Next, we explain visibility, overlap and continuity w.r.t. portions.

$vis(q_i, p_j)$ : Given a query location $q_i$, we consider $vis(q_i, p_j) = true$ if the entire portion $p_j$ is visible from $q_i$; otherwise, $vis(q_i, p_j)$ is $false$.

$VR(q_i, TP)$ : The visible region $VR(q_i, TP)$ of $TP$ from $q_i$ is the union of all the portions of $TP$ that are visible from $q_i$. Hence, $VR(q_i, TP) = \{p_j | p_j \in P \& vis(q_i, p_j) = true\}$. To compute $VR(q_i, TP)$ $\forall q_i \in Q$, we apply the algorithm proposed in [18] for computing the VRs in terms of portions. The number of portions in $VR(q_i, TP)$ is impacted by obstacles. Only the obstacles, which obstruct the line of sight from $q_i$ to $p_j$, impact $vis(q_i, p_j)$ where $p_i \in P$. In the approach in [18], all the obstacles are indexed using an R*-tree [3]. The obstacles, which fall in the line of sight of $p_j$ and $q_i$, are extracted from the R*-tree in the order of nearest distance to $p_j$. This is because the nearest obstacle has maximum impact on the visibility, thereby facilitating the efficient determination of $VR(q_i)$.

$vis(c_i, TP)$ : The visibility $vis(c_i, TP)$ of candidate set $c_i$ consists of $n_i$ query locations is the ratio of number of portions visible from all of the query locations in set $c_i$ to the total number of portions of the target object $TP$.

$$vis(c_i, T) = |\bigcup_{i=1}^{n_i}(VR(q_i, T))|/|P| \quad (4)$$

The range of $vis(c_i, TP)$ is 0 to 1. Recall that $minV$ in Section 2 represents the user-specified minimum visibility threshold. For $c_i$ to be the candidate set of MLV query, $vis(c_i, TP) \geq minV$.

$overlap(c_i, T)$ : Let the VR of a candidate set $c_i$ of query locations be $VR(c_i, T)$. Suppose we want to capture more visibility of $TP$ by adding a new query location $q_j$ to $c_i$. Let the VR of $q_j$ be $VR(q_j, TP)$. Notably, adding $q_j$ to $c_i$ could result in overlap between $VR(c_i, TP)$ and $VR(q_j, TP)$. If the overlap is high, $vis(c_i \cup q_j, TP)$ may not increase significantly over $vis(c_i, TP)$. Hence, from a visibility perspective, adding $q_j$ to $c_i$ would not be interesting. In essence, adding a new query location $q_j$ to set $c_i$ would be interesting only if there is a minimal overlap between the visible regions of $c_i$ and $q_j$. The overlap $overlap(c_i, TP)$ of $c_i$ is defined as follows. Let $c_i = c_j \cup q_k$ . Then, $overlap(c_i, TP)$ is equal to the ratio of the number of portions common in $VR(c_j, TP)$ and $VR(q_k, TP)$ to the number of portions in $VR(q_k, TP)$.

$$overlap(c_i, TP) = (|VR(c_j, TP) \cap VR(q_k), TP|)/|VR(q_k, TP|) \quad (5)$$

The range of $overlap(c_i, T)$ is 0 to 1. Recall that $maxO$ represents the user-specified maximum overlap threshold. For $c_i$ to be the candidate set of CMLV query, $overlap(c_i, T) \leq maxO$.

Regarding continuity among portions, given $p_i$, $p_j$ from the set $P$, $p_i$ and $p_j$ form a one-to-one continuity relationship iff $sp(p_i) = ep(p_j)$ or $sp(p_j) = ep(p_i)$. We consider $p_i$ and $p_j$ to be continuous if there is a transitive-closure of one-to-one continuity relationship between $p_i$ and $p_j$ over $P$. We designate the continuity for candidate sets $c_i$ as $continuity(c_i, TP)$. Thus, $continuity(c_i, TP)$ is $true$ iff for every pair of portions $\{p_i, p_j\} \in VR(c_i, TP)$ are continuous. Otherwise, $continuity(c_i, T)$ is $false$.

### 4.2.2 Pre-computation of portion transactions

**Portion Transactions** : Consider a set $Q$ of $n$ query locations $\{q_i, q_j ...., q_n\}$ and a set $P$ of $w$ portions $\{p_i, p_j, ...., p_w\}$ of $TP$. $S$ is the set of portion transactions, where each portion transaction $pt_i$ for portion $p_i$ is a set $Q'$=$\{q_j |\ q_j \in Q'$

and $vis(p_i, q_j)=true\}$. Therefore, portion transaction $pt_i$ is of the form $\langle pid_i, Q' \rangle$, where $pid_i$ is the identifier of $pt_i$.

Notably, the number of portion transactions formed is equal to the number of portions.

Algorithm 2 depicts the generation of portion transactions of target's path. The inputs are set $P$ of portions of $TP$, set $O$ of obstacles and set $Q$ of query locations. For generating a set $S$ of portion transactions, we extract $VR(q_i, TP)$ for each $q_i \in Q$. We form transactions of the form $\langle pid_i, Q' \rangle$ by associating $p_i$ and $Q'$, where $Q'$ contains all $q_i$ such that $vis(q_i, p_i) = true$. For each portion with a portion id $pid_k$ in $VR(q_i, TP)$ of each query location $q_i$ , $q_i$ is added to the portion transaction with id as $pid_k$.

**Algorithm 2:** Generating Portion Transactions of Target's path

**Input :** Set $P$ of portions, set $Q$ of query locations, set $O$ of obstacles
**Output:** Set $S$ of Portion transactions
Initialize $S$ to empty set;
Compute $VRs$ w.r.t. portions;
1: **for** each $VR(i)$ in $VR$ **do**
2: **for** each portion $pid$ in $VR(i)$ **do**
3: $S[pid].append(q_i)$
4: **end for**
5: **end for**
return $S$;

### 4.2.3 Finding portion transactions of target T

To find top-$k$ candidate sets such that we can maximize the visibility of the target, we need to have portion transactions of target $T$ not of target path $TP$. To generate portion transactions at every time instant we should consider all obstacles and query points at every time instant which leads to high computation overhead. To overcome this issue we will compute portion transactions of target path only once considering all obstacles and query points and from these portion transactions of target path we generate portion transactions of the target. We denote $\langle S(T), E(T) \rangle$ as the starting and ending point of the target, where $S(T) = (x_i, y_i)$ and $E(T) = (x_j, y_j)$.

Algorithm 3 depicts the finding the portion transactions of the target from the portion transactions from the target path. The inputs are set $P$ of portions of the target path and the location $l_i$ at time instant $t_i$. For generating portion transactions of target we find which part of the portions of the path are overlapping with the target and adding those respective overlapping portion transactions of the path to the target portion transactions at location $l_i$.

### 4.2.4 Determining the top-k candidate sets incrementally

At the first instant of time we calculate from top-$k$ candidate sets by the traditional approach in [6]. From the next instant onwards if we again mine the portion transactions for every instant to generate top-$k$ candidate sets would incur high computational cost and execution time. To overcome this issue we proposed an incremental algorithm which uses the previous information of candidate sets as well as the portion transactions. The idea behind this technique is when we compare the portion transactions of the previous instant

**Algorithm 3:** Finding Portion Transactions of Target

**Input :** Set of portions $P$, location of Target object $T$
**Output:** Set $PTT$ of Portion transactions of target
Initialize $PTT$ to empty set;
location of $T \leftarrow \langle S(T), E(T) \rangle$;
portion in $TP \leftarrow \langle sp(s), ep(s) \rangle$;
$sp(s), ep(s)$ are start and end-points of portion;
1: **for** each portion $\langle sp(s), ep(s) \rangle$ in $P$ **do**
2: **if** $sp(s) \leq S(T).x \leq ep(s)$ **then**
3: Id of starting portion = $pid_i$
4: **end if**
5: **if** $sp(s) \leq E(T).x \leq ep(s)$ **then**
6: Id of ending portion = $pid_j$
7: **end if**
8: $PTT = PTT \bigcup_{i=pid_i}^{pid_j} S_k$
9: **end for**
return $PTT$;

and the current instant we can observe that there are many common transactions. So to generate top-$k$ candidate sets for every instant we use previous candidate sets, portion transactions of the previous time instant and portion transactions of the current time instant.

[Ask sir how to write this algorithm]

# 5. PERFORMANCE EVALUATION

# 6. DISCUSSION

# 7. CONCLUSION

Visibility computation is important in spatial databases for realizing various applications. While existing works address the problem of finding the identifying individual query locations for maximizing the visibility of a moving target object $T$, we consider the problem of determining multiple query locations for maximizing the visibility of a moving target $T$. We have introduced $CMLV$ query and proposed an efficient pre-processing and incremental approach to process $CMLV$ queries. Our performance evaluation with real datasets demonstrates the effectiveness of the proposed scheme in terms of query processing time and target object visibility w.r.t. a recent existing scheme. As visibility computation is an active research area, due to efficiency and flexibility, the proposed approach provides the scope to explore efficient approaches for improving robotic and drone based surveillance applications.

# 8. REFERENCES


[1] D. Ali, M. Eunus, et al. Efficient processing of maximum visibility facility selection query in spatial databases, Masters Thesis, BUET. 2017.

[2] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1):49–63, 1986.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access

method for points and rectangles. In *Proc. ACM SIGMOD*, pages 322–331, 1990.

[4] V. Chvatal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, 1979.

[5] L. Floriani and P. Magillo. Algorithms for visibility computation on terrains: a survey. *Environment and Planning B: Planning and Design*, 30(5):709–728, 2003.

[6] L. Gangumalla, P. K. Reddy, and A. Mondal. Multi-location visibility query processing using portion-based transactional modeling and pattern mining. *Data Mining and Knowledge Discovery*, 33(5):1393–1416, 2019.

[7] Y. Gao, Q. Liu, X. Miao, and J. Yang. Reverse k-nearest neighbor search in the presence of obstacles. *Information Sciences*, 330:274–292, 10 2015.

[8] Y. Gao and B. Zheng. Continuous obstructed nearest neighbor queries in spatial databases. In *Proc. ACM SIGMOD*, pages 577–590, 2009.

[9] Y. Gao, B. Zheng, G. Chen, W.-C. Lee, K. C. K. Lee, and Q. Li. Visible reverse k-nearest neighbor queries. In *Proc. ICDE*, pages 1203–1206, 2009.

[10] Y. Gao, B. Zheng, W.-C. Lee, and G. Chen. Continuous visible nearest neighbor queries. In *Proc. EDBT*, pages 144–155, 2009.

[11] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. In *Proc. ACM Symposium on Theory of Computing*, pages 47–63, 1974.

[12] M. F. Goodchild and J. Lee. Coverage problems and visibility regions on topographic surfaces. *Ann. Oper. Res.*, 18(1-4):175–186, 1990.

[13] e. a. Gowtham Srinivas, P. Mining coverage patterns from transactional databases. *J. Intell. Inf. Syst.*, 45(3):423–439, 2015.

[14] C. M. R. Haider, A. Arman, M. E. Ali, and F. M. Choudhury. Continuous maximum visibility query for a moving target. In *Australasian Database Conference*, pages 82–94. Springer, 2016.

[15] C. M. R. Haider, A. Arman, M. E. Ali, and F. M. Choudhury. Continuous maximum visibility query for a moving target. In *Proc. ADC*, pages 82–94, 2016.

[16] N. Irtiza Tripto, M. Nahar, M. E. Ali, F. Choudhury, J. Culpepper, and T. Sellis. Top-k trajectories with the best view. *GeoInformatica*, page 1–41, 02 2019.

[17] Y. Li, J. Yang, and J. Han. Continuous k-nearest neighbor search for moving objects. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pages 123–126. IEEE, 2004.

[18] S. Masud, F. M. Choudhury, M. E. Ali, and S. Nutanong. Maximum visibility queries in spatial databases. In *Proc. ICDE*, pages 637–648, 2013.

[19] K. Morling. *Geometric and Engineering Drawing 3E.* Routledge, 2010.

[20] S. Nutanong, E. Tanin, and R. Zhang. Incremental evaluation of visible nearest neighbor queries. *IEEE TKDE*, 22(5):665–681, 2010.

[21] M. A. Seeds and D. Backman. *Stars and galaxies.* Cengage Learning, 2015.

[22] L. Shou, Z. Huang, and K.-L. Tan. HDoV-tree: The structure, the storage, the speed. In *Proc. ICDE*, pages 557–568, 2003.

[23] P. G. Srinivas and et. al. Discovering coverage patterns for banner advertisement placement. In *Proc. PAKDD*, pages 133–144, 2012.

[24] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proc. VLDB*, pages 287–298, 2002.

[25] H. Xu, Z. Li, Y. Lu, K. Deng, and X. Zhou. Group visible nearest neighbor queries in spatial databases. In *Proc. WAIM*, pages 333–344, 2010.