Towards Efficient Maintenance of Continuous MaxRS Query for Trajectories

Muhammed Mas-ud Hussain[1] Goce Trajcevski[*] [1]
Kazi Ashik Islam[2] Mohammed Eunus Ali[2]

[1]Department of Electrical Engineering and Computer Science
Northwestern University, Evanston, IL, 60208
{mmh683, goce@eecs.northwestern.edu}

[2] Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
{1205007.kai@ugrad.cse.buet.ac.bd, eunus@cse.buet.ac.bd}

ABSTRACT

We address the problem of efficient maintenance of the answer to a new type of query: Continuous Maximizing Range-Sum (Co-MaxRS) for moving objects trajectories. The traditional static/spatial MaxRS problem finds a location for placing the centroid of a given (axes-parallel) rectangle R so that the sum of the weights of the point-objects from a given set O inside the interior of R is maximized. However, moving objects continuously change their locations over time, so the MaxRS solution for a particular time instant need not be a solution at another time instant. In this paper, we devise the conditions under which a particular MaxRS solution may cease to be valid and a new optimal location for the query-rectangle R is needed. More specifically, we solve the problem of maintaining the trajectory of the centroid of R. In addition, we propose efficient pruning strategies (and corresponding data structures) to speed-up the process of maintaining the accuracy of the Co-MaxRS solution. We prove the correctness of our approach and present experimental evaluations over both real and synthetic datasets, demonstrating the benefits of the proposed methods.

1. INTRODUCTION

Recent technological advances in miniaturization of position-aware devices equipped with various sensors, along with the advances in networking and communications, have enabled a generation of large quantities of *(location, time)* data – O(Exabyte) [16]. This, in turn, promoted various geo-social applications where the *(location, time)* information is enriched with (sensed) values from multiple contexts [30, 31]. At the core of many such applications of high societal relevance – e.g., tracking in ecology and environmental monitoring, traffic management, online/targeted marketing, etc. – is the efficient management of mobility data [23].

Researchers in the Spatio-temporal [15] and Moving Objects Databases (MOD) [9] communities have developed a plethora of methods for efficient storage and retrieval of the whereabouts-in-time data, and efficient processing of various queries of interest. Many of those queries – e.g., range, (k) nearest neighbor, reverse nearest-neighbor, skyline, etc. – have had their "predecessors" in traditional relational database settings, as well as in spatial databases [27]. However, due to the motion, their spatio-temporal variants became continuous (i.e., the answer-sets change over time) and even persistent (i.e., answers change over time, but also depend on the history of the motion) [20, 32].

In a similar spirit, this work explores the spatio-temporal extension of a particular type of a spatial query – the, so called, Maximizing Range-Sum query (MaxRS), which can be described as follows:

Q: *"Given a collection of weighted spatial point-objects O and a rectangle R with fixed dimensions, finds the location(s) of R that maximizes the sum of the weights of the objects in R's interior".*

Various aspects of MaxRS (e.g., scalability, approximate solutions, insertion/removal of points) have been addressed in spatial settings [5, 7, 11, 22, 25, 28] – however, our main

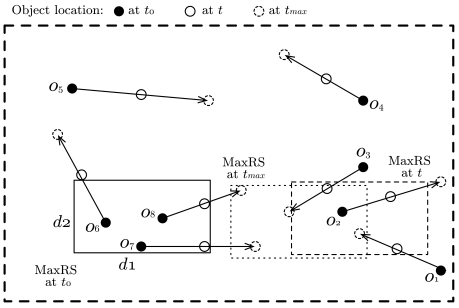


Figure 1: MaxRS vs. Co-MaxRS.

*Research supported by NSF grants III 1213038 and CNS 1646107, and the ONR grant N00014-14-10215.

motivation is based on the observation that there are many application scenarios for which efficient processing of the *continuous* variant of MaxRS is essential. Consider the following query:

Q1: *"What should be the trajectory of a drone which ensures that the number of mobile objects in the Field-of-View of its camera is always maximal?"*.

It is not hard to adapt **Q1** to other application settings: – environmental tracking (e.g., optimizing a range-bounded continuous monitoring of a herd of animals with highest density inside the region); – traffic monitoring (e.g., detecting ranges with densest traffic between noon and 6PM); – video-games (e.g., determining a position of maximal coverage in dynamic scenarios involving change of locations of players/tanks in World of Tanks game). Pretty much any domain involving continuous detection of "most interesting" regions involving mobile entities is likely to benefit from the efficient processing of variants of **Q1** (e.g., mining popular trajectories patterns [33], sports analytics [26], etc.).

Contrary to the traditional *range* query which detects the number of points, or higher dimensionality objects such as (poly)lines and shapes, related to a given *fixed* region, the MaxRS determines the location for placing a given region so that the sum of the weights (i.e., some objective function related to location) is maximized. Originally, the MaxRS problem was tackled by the researchers in computational geometry [11, 22] – however, motivated by its importance in LBS-applications – e.g., best location for a new franchise store with a limited delivery range, most attractive place for a tourist with a restricted reachability bound – recent works have proposed scalable efficient solution for MaxRS in spatial databases [5], including approximate solutions [28] and scenarios where the weights may change and points may be added/deleted [7].

However, the existing solutions to MaxRS queries can only be applied to a specific time instant – whereas **Q1** is a *Continuous MaxRS* (Co-MaxRS) variant. Its weighted-version would correspond to prioritizing certain kinds of mobile objects (e.g., areas with most trucks – by assigning higher weights to trucks) to be tracked by the drone, or certain kinds of tanks in the World of Tanks game. The fundamental difference between MaxRS and Co-MaxRS is illustrated in Figure 1. Assuming that the 8 objects are static at time t_0 and the weights of all the objects are uniform, the placement of the rectangle R indicated in solid line is the solution, i.e., count for optimal R is 3. Other suboptimal placements are possible too at t_0, e.g., covering only o_2 and o_3 with count being 2. However, when objects are mobile, the placement of R at different time instants may need to be changed – as shown in Figure 1 for t_0, t and t_{max}.

A few recent works have tackled the dynamic variants of the MaxRS problem [1, 21]. These works consider objects that may appear or disappear (i.e., insert/delete); however, the locations of the objects do not change over time. To the best of our knowledge, the Co-MaxRS problem has not been addressed in the literature so far and the main contribution of our work can be summarized as follows:

- We formally define the Co-MaxRS problem and identify criteria (i.e., *critical times*) under which a particular MaxRS solution may no longer be valid, or a new MaxRS solution emerges. These, in turn, enable algorithmic solution to Co-MaxRS using procedures executing at discrete time instants.
- Given the worst-case complexity of the problem (consequently, the algorithmic solution), we propose efficient pruning strategies to reduce the cost of recomputing the Co-MaxRS solutions at certain critical times. We present an in-memory data structure and identify properties that enable two such strategies: (1) eliminating the recomputation altogether at corresponding critical time; (2) reducing the number of objects that need to be considered when recomputing the Co-MaxRS solution at given critical times.
- We evaluate our proposed approaches using both real and synthetic datasets, and demonstrate that the pruning strategies yield much better performance than the worst-case theoretical bounds of the Co-MaxRS algorithm – e.g., we can eliminate 80-90% of the critical time events and prune ∼70% objects (on average) when recomputing Co-MaxRS.

In the rest of this paper, Section 2 presents the basic technical background, and Section 3 formalizes the Co-MaxRS problem and describes the basic properties and algorithmic aspects of its solution. Section 4 presents the details of our pruning strategies: properties, data structures and algorithms, and Section 5 presents the quantitative experimental observations illustrating the benefits of the proposed pruning. Section 6 positions the work with respect to the related literature, and Section 7 offers conclusions and directions for future work.

2. PRELIMINARIES

We now review the approaches for solving static MaxRS problem and introduce the concept of kinetic data structures that we subsequently use for solving Co-MaxRS.

2.1 MaxRS for Static Objects

Let $C(p, R)$ denote the region covered by an isothetic rectangle R, placed at a particular point p. Formally:

Definition 1. **(MaxRS)** Given a set O of n spatial points $O = \{o_1, o_2, \ldots, o_n\}$, where each o_i associated with[1] a weight w_i , the answer to MaxRS query ($\mathbb{A}_{MaxRS}(O, R)$) retrieves a position p for placing the center of R, such that $\sum_{\{o_i \in (\ O \cap C(p,R))\}} w_i$ is maximal.

$\sum_{\{o_i \in (\ O \cap C(p,R))\}} w_i$ is called the *score* of R located at p. If $\forall o_i \in O : w_i = 1$, we have the *count* variant, instances of which at different times are shown in Figure 1. Note that there may be multiple solutions to the MaxRS problem, and in the case of ties – one can be chosen randomly, unless other ranking/preference criteria exist.

Consider the example shown in Figure 2 – the count variant of MaxRS, with a rectangle R of size $d_1 \times d_2$ and five objects (black-filled circles). An in-memory solution to MaxRS (cf. [22]) transforms it into a "dual" *rectangle intersection problem* by replacing each object in $o_i \in O$ by a $d_1 \times d_2$ rectangle r_i, centered at o_i. R covers o_i if and only if its center is placed within r_i. Thus, the rectangle covering the maximum number of objects can be centered anywhere within the area containing a maximal number of intersecting dual rectangles (e.g., $r_3 \cap r_4 \cap r_5$ – gray-filled area in Figure 2).

Using this transformation, an in-memory algorithm to solve the MaxRS problem in $O(n \log n)$ time and $O(n)$ space was devised in [22]. Viewing the top and the bottom edges of each rectangle as horizontal intervals, an *interval tree* – i.e., a binary tree on the intervals – is constructed, and then a

[1]One may also assume that the points in O are bounded within a rectangular area $\mathbb{F}$.

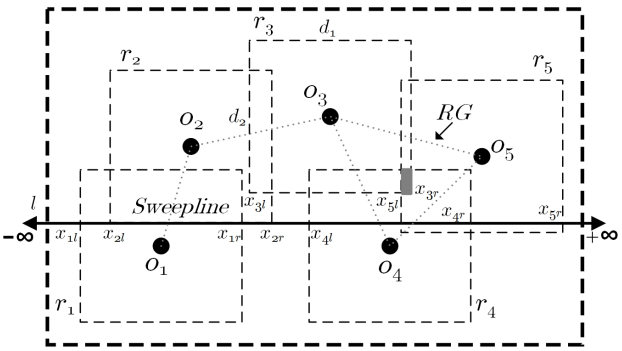


Figure 2: MaxRS → rectangle intersection.

horizontal line is swept vertically, updating the tree at each event. The algorithm maintains the count for each interval currently residing in the tree, where the count of an interval represents the number of overlapping rectangles within that interval. When the sweep-line meets the bottom (top) edge of a rectangle, the corresponding interval is inserted to (deleted from) the interval tree and the count of each interval is updated accordingly. Considering the scenario in Figure 2 and using $[x_{il}, x_{ir}]$ to denote the left and right boundaries of r_i, when the horizontal sweep-line is at position l, there are 9 intervals: $[-\infty, x_{1l}]$, $[x_{1l}, x_{2l}]$, $[x_{2l}, x_{1r}]$, $[x_{1r}, x_{2r}]$, $[x_{2r}, x_{4l}]$, $[x_{4l}, x_{5l}]$, $[x_{5l}, x_{4r}]$, $[x_{4r}, x_{5r}]$, and $[x_{5r}, +\infty]$—with counts of 0, 1, 2, 1, 0, 1, 2, 1, and 0 respectively. An interval with the maximum count during the entire sweeping process is returned as the final solution and, since there can be at most $2n$ events (top or bottom horizontal edge of all r_i's) and each event takes $O(\log n)$ processing time, the whole algorithm takes $O(n \log n)$ time to complete.

We note that one may construct a graph RG (*rectangle graph*) where vertices correspond to points/objects in O (i.e., the centers of the dual rectangles) and an edge exists between two vertices o_i and o_j if and only if the corresponding dual rectangles overlap (i.e., $r_i \cap r_j \neq \emptyset$). As illustrated with dotted edges in Figure 2, an area of maximum overlap of dual rectangles corresponds to a maximum clique in RG.

2.2 Kinetic Data Structures

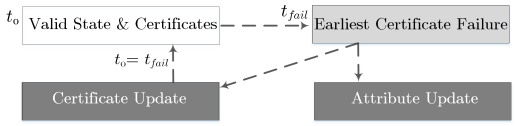


Figure 3: Kinetic Data Structures paradigm.

Kinetic data structures (KDS) [2] are used to track attributes of interest in a geometric system, where there is a set of values (e.g., location – x and y coordinates) that are changing as a function of time in a known manner. To process queries at a (virtual) current time t, an instance of the data structure at initial time t_0 is stored (i.e., values of the attributes of interest), which is augmented with a set of *certificates* proving its correctness at t_0. The next step is to compute the failure times of each certificates – called *events* – indicating that the data structure may no longer be an accurate representation of the state of the system. The events are stored in a priority queue sorted by their failure times. To advance to a time t $(= t_0 + \delta)$, we have to pop all the events having failure times $t_{fail} \leq t_0 + \delta$ from the queue in-order, and perform two operations at each event: (1) modify the data structure so that it is accurate at t_{fail} (attribute update), and (2) update the related certificates accordingly (see Figure 3). In this paper, we utilize KDS to maintain the Co-MaxRS answer-set over time and only perform certain tasks at the critical times (events) when a current MaxRS solution may change.

3. BASIC CO-MAXRS

Interval tree was used as in-memory data structure of the planesweep algorithm in both [22] and the subsequent work addressing scalability [5]. However, these techniques cannot be straightforwardly extended to maintain MaxRS solutions continuously – i.e., one cannot expect to have an uncountably-infinite amount of interval trees (at each instant of objects' motion). As it turns out, the answer to Co-MaxRS can change only at discrete time-instants, which we address in the sequel.

Throughout this section, without loss of generality, we assume that each object moves along a single straight line-segment and all the objects start and finish their motion in the same time instant. We will lift this assumption and discuss its impact in Section 4.3.

Continuous MaxRS (Co-MaxRS) is defined as follows:

Definition 2. **(Co-MaxRS)** Given a set O_m of n 2D moving points $O_m = \{o_1, o_2, \ldots, o_n\}$, where each is associated with a trajectory[2] $o_i = [(x_{i1}, y_{i1}, t_{i1}), \ldots, (x_{i(k+1)}, y_{i(k+1)}, t_{i(k+1)})]$ and a weight w_i; and a time-interval $T = [t_0, t_{max}]$, the answer to Co-MaxRS ($\mathbb{A}_{Co\text{-}MaxRS}(O_m, R, T)$) is a (time-ordered) sequence of pairs $[(l^1_{obj}, [t_0, t_1)), (l^2_{obj}, [t_1, t_2)), \ldots, (l^c_{obj}, [t_{c-1}, t_{max}))]$, where $(l^i_{obj}, [t_{i-1}, t_i))$ denotes the set of objects that determine the possible location(s) for R that is a MaxRS at any time instant $t_j \in [t_{i-1}, t_i) (\subseteq T)$.

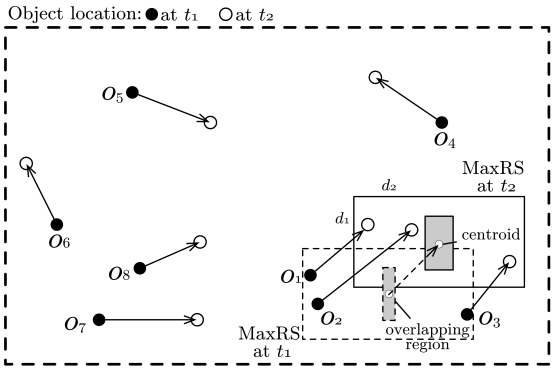


Figure 4: MaxRS location changes from t_1 to t_2, although the objects in the solution are the same.

Note that, instead of maintaining a centroid-location (equivalently, a region) as a Co-MaxRS solution, we maintain a list of objects that are located in the interior of the optimal rectangle placement. The rationale is two-fold: (1)

[2]Again, the trajectories may be bounded within a rectangular area $\mathbb{F}$.

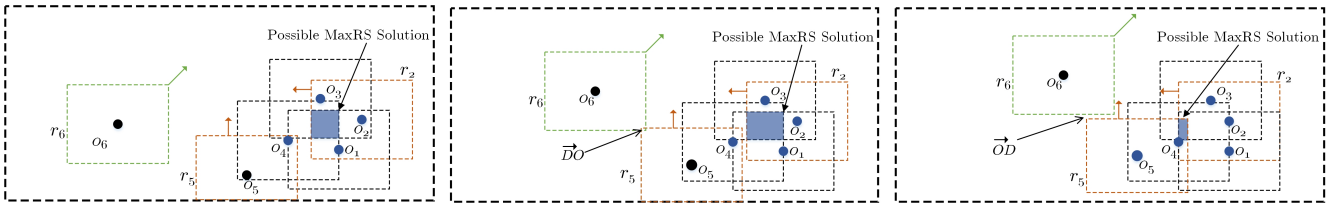


Figure 5: Co-MaxRS answer can only change when two rectangles' relationship changes from overlap to disjoint (or, vice-versa). Object locations at: (a) t_1 (b) t_2 (c) t_3.

Even for small object movements, the optimal location of the query rectangle can change while objects participating in the MaxRS solution stay the same; and (2) We can easily determine the trajectory (one of the uncountably-many) of the centroid of R throughout the time-interval during which the same set of objects constitutes the solution. An example is shown in Figure 4. At time t_1, objects o_1, o_2, and o_3 fall in the interior of the MaxRS solution. At t_2, although the same objects constitute the MaxRS solution, the optimal location itself has shifted due to the movement of the objects. Suppose there are s objects in the list l^j_{obj} at a particular time instant $t_s \in [t_{j-1}, t_j)$. Given l^j_{obj}, one can find the intersection of the s dual rectangles to retrieve the (boundaries of the possible) location for R at t_s in $O(s)$ time.

We can readily consider an alternative way of representing the Co-MaxRS solution – namely, as a trajectory of the (placement of the) centroid of R. Consider any time interval during which the same set of objects constitutes the solution – e.g., again $(l^j_{obj}, [t_{j-1}, t_j))$. Let $\{o^j_1, \ldots, o^j_s\}$ denote the actual objects from O_m defining l^j_{obj}. Their respective dual rectangles, $\{r^j_1, \ldots, r^j_s\}$ have a common intersecting region at $t = t_{j-1}$ – which, by assumption, is an axes-parallel rectangle. Every point in $\cap_{i=1}^{i=s} r^j_i$ can be a centroid of R covering l^j_{obj} at t_{j-1}. Similarly for $t = t_j$ – once again we have an intersection of the s objects yielding an axes-parallel rectangle, except that both its size and location are changed with respect to the one at $t = t_{j-1}$. The key observations are:
(1) Each r^j_i dual rectangle, when moving along a straight line-segment (to follow the o^j_i) between t_{j-1} and t_j, "swipes" a volume corresponding to a sheared box/parallelopiped.
(2) At each $t \in [t_{j-1}, t_j)$ the intersection of the dual rectangles is non-empty (otherwise, it would contradict the fact that the objects in l^j_{obj} define the solution) and is a rectangle, thereby ensuring that the intersection of the parallelopipeds is continuously non-empty and, again, convex.

Thus, given the $\mathbb{A}_{MaxRS}(O, R)$ at $t = t_{j-1}$ and $t = t_j$, we can simply pick a point in the interior of each of the two (horizontal) rectangles in the $(X, Y, Time)$ space, and the line-segment connecting them is one of the possible trajectories of the centroid of R as the solution/answer-set $\mathbb{A}_{Co\text{-}MaxRS}(O, R, T)$ (of course, for $T = [t_{j-1}, t_j)$).

We now describe how to identify when a recomputation of the MaxRS may (not) be needed due to the possibility of a change in the solution. Consider the example in Figure 5 with 6 objects: $\{o_1, o_2, \ldots, o_6\}$. Let r_i denote the dual rectangle for an object o_i. For simplicity of visualization, assume that only o_2, o_5 and o_6 are moving: o_2 in west, o_5 in north direction (orange rectangles and arrows), and o_6 in the northwest direction. Figure 5a, shows the locations of objects at t_1 and the current MaxRS solution, $l_{obj} = \{o_1, o_2, o_3, o_4\}$ (blue colored objects in Figure 5a). In this setting, r_2 and r_5 do not overlap. Figure 5b shows the objects' locations and their corresponding rectangles at t_2 ($> t_1$). Due to the movement of o_2 and o_5, the maximum overlapped area changed at t_2 (blue-shaded region). But, as r_2 and r_5 still do not overlap, the objects comprising the MaxRS solution are still the same as t_1. Finally, Figure 5c represents the objects' locations at a later time t_3, where r_2 and r_5 are overlapping. This causes a change in the list of objects making up the MaxRS solution, and o_5 is added to the current solution. We note that the solution changed only when two disjoint rectangles began to overlap. If we consider the example in reverse temporal order, i.e., assuming $t_3 < t_2 < t_1$, then the MaxRS solution changed when two overlapping rectangles became disjoint.

Observation: The solution of Co-MaxRS changes only when two rectangles change their topological relationship from *disjoint* to *overlapping* ($\vec{DO}$), or from *overlapping* to *disjoint* ($\vec{OD}$). We consider the objects along the boundary of the query rectangle R as being in its interior, i.e., rectangles having partially overlapping sides and/or overlapping vertices are considered to be *overlapping*. In the rest of the paper, if we need to indicate an occurrence of $\vec{DO}$ or $\vec{OD}$ at a specific time instant t and pertaining to two specific objects o_i and o_j, we will extend the signature of the notation by adding time as a parameter and index the objects in the subscript (e.g., $\vec{DO}_{i,j}(t)$ or $\vec{OD}_{i,j}(t)$).

Thus, as the objects (resp. dual rectangles) move, there are two kinds of changes:
(1) *Continuous Deformation:* As the locations of the rectangles change, the overlapping rectangular regions may change, but the set of objects determining any overlapping rectangular region remains the same.
(2) *Topological Change:* Due to the movement of the rectangles, a $\vec{DO}$ or $\vec{OD}$ transition occurs for a pair of rectangles.

We note that, while the change of the topological relationship is necessary for a change in the answer set in the continuous variant of $\mathbb{A}_{MaxRS}(O_m, R)$ – it need not be sufficient. As shown in Figure 5, the relationship between r_5 and r_6 transitioned from disjoint, to overlap, and to disjoint again. However, none of those changes affected the $\mathbb{A}_{Co\text{-}MaxRS}(O_m, R, T)$ between t_1 and t_3.

In Section 4.3 we will use this observation when investigating the options of pruning certain events corresponding to changes in topological relationships. At the time being, we summarize the steps for a brute-force algorithm for calculating the answer to Co-MaxRS:

Algorithm 1 Basic Co-MaxRS

Input: $(O_m, R, T = [t_0, t_{max}])$

1: Calculate all the time instants for all the pairwise topological changes for the objects in O_m
2: Sort the times of topological changes
3: For each such time t_i^{tc}, execute $\mathbb{A}_{MaxRS}(O, R)$
4: **if** Objects defining the answer set are the same **then**
5: Extend the time-interval of the validity of the most recent entry in $\mathbb{A}_{Co\text{-}MaxRS}$ $(O_m, R, T = [t_0, t_{max}])$
6: **else**
7: Close the time-interval of validity of the prior most-recent entry
8: Add a new element into $\mathbb{A}_{Co\text{-}MaxRS}$ $(O_m, R, T = [t_0, t_{max}])$ consisting of the objects defining the $\mathbb{A}_{MaxRS}(O, R)$ at t_i^{tc}, with the interval $[t_i^{tc}, t_{i+1}^{tc})$
9: **end if**
10: **return** $\mathbb{A}_{Co\text{-}MaxRS}(O_m, R, T)$

Clearly, the complexity of Algorithm 1 is $O(n^3 \log n)$ – which can be broken into: – $O(n^2)$ for determining the (pairwise) times of topological changes; – $O(n^2 \log n^2)$ for sorting those times; – executing $O(n^2)$ times the instantaneous $\mathbb{A}_{MaxRS}(O, R)$ (at $O(n \log n)$). We note that $O(n^3 \log n)$ is actually a tight worst-case upper-bound, since the solutions in $\mathbb{A}_{MaxRS}(O, R)$ can be "jumping" from one R-region into another that is located elsewhere in the area of interest between any two successive intervals – which are $O(n^2)$.

4. PRUNING IN CO-MAXRS

Given the complexity of the naïve solution – which, again, captures the worst-case possible behavior of moving objects – we now focus on strategies that could reduce certain computational overheads, based on (possible) "localities". We discuss two such strategies aiming to: (1) Reduce the number of recomputations of MaxRS; and (2) Reduce the total number of objects considered when recomputing the MaxRS solution[3], and then present the algorithms that exploit those strategies.

Before proceeding with the details of the pruning, we describe the data structures used.

Figure 6 depicts the data structures used to maintain the Co-MaxRS answer-set based on the KDS framework. Strictly speaking, it consists of:

Object List (OL): A list for storing each object $o_i \in O$, with its current trajectory Tr_{o_i} (i.e., snapshots of location at t_0 and t_{max}), weight w_i, sum of weights of its neighbors in the rectangle graph $WN(o_i)$, and whether or not the object is part of the current MaxRS solution. Note that, o_j is neighbor of o_i if r_i and r_j overlap.

Kinetic Data Structure (KDS): Figure 6 illustrates the underlying KDS (event queue), and its relation with the OL. Each event $E_{i,j}^{t_k}$ is associated with a time t_k, where $t_0 < t_k < t_{max}$. KDS maintains an event queue, where the events are sorted according to the time-value. Each event entry $E_{i,j}^{t_k}$ has pointers to its related objects – two object *identifiers*, and the type of the event – ($\vec{D}O$ or $\vec{O}D$).

Adjacency Matrix (AdjMatrix): Represents the time-dependent rectangle graph RG, with its rows and columns corresponding to the vertices of RG (i.e., the objects from O_m). For each pair of objects o_i and o_j, and a particular (critical) time instant, the $AdjMatrix[i][j]$ and $AdjMatrix[j][i]$ – set to 1 or 0 – indicate whether the two objects are directly connected with an edge in RG (i.e., their dual rectangles overlap).

[3]Due to a lack of space, we do not present the proofs of the Lemmas in this paper, however, they are available at [17].

4.1 Pruning KDS Events

Recall that the solution to MaxRS problem is equivalent to retrieving the maximum clique in the rectangle graph RG (cf. Section 2). For our first kind of pruning methodology, we leverage on the fact that a KDS event involving two objects o_i and o_j – which can be either $\vec{D}O_{ij}$ or $\vec{O}D_{ij}$ – is equivalent to adding or deleting an edge only between r_i and r_j, and no other objects/rectangles are involved. The properties that allow us to filter out $\vec{D}O$ and/or $\vec{O}D$ types of events without recomputing the MaxRS are discussed next.

$\vec{D}O$: Let $WN(o_i)(t)$ denote the current sum of the weights of the neighbors of an object o_i at time t, and let $score_{max}(t) = score((\mathbb{A}_{MaxRS}(O, R)), t)$ denote the score of the current MaxRS solution at t. During a ***$\vec{D}O$*** event, the lower bound of a MaxRS solution is $score_{max}(t)$, and upper bound of the score (i.e., maximum possible score) of an overlapping region including an object o_i is $(WN(o_i) + w_i)$.

LEMMA 1. *Consider the event $\vec{D}O_{i,j}$ for two objects o_i and o_j, occurring at time $t_{i,j}$. Let $l_{obj}^{(t_{i,j}-\delta)}$ (for some small δ) denote the Co-MaxRS solution just before $t_{i,j}$. After updating $WN(o_i)$ and $WN(o_j)$ at $t_{i,j}$ (i.e., because of $\vec{D}O_{i,j}$), $l_{obj}^{(t_{i,j}-\delta)}$ remains a MaxRS if one of the following two inequalities holds:*

(1) $WN(o_i)(t_{i,j}) + w_i \leq score_{max}(t_{i,j} - \delta)$
(2) $WN(o_j)(t_{i,j}) + w_j \leq score_{max}(t_{i,j} - \delta)$

$\vec{O}D$: In this case the intuition is much simpler – the score/count of an instantaneous MaxRS solution can only decrease (or, remain same) during an $\vec{O}D$ event, and if it decreases (i.e., changes), both of the objects involved in the event must have been in l_{obj}. Thus, we have:

LEMMA 2. *Consider the event $\vec{O}D_{ij}$ for two objects o_i and o_j occurring at time $t_{i,j}$. Let $l_{obj}^{(t_{i,j}-\delta)}$ (for some small δ) be the current MaxRS solution before $t_{i,j}$. If one of the following two conditions holds:*

(1) $o_i \notin l_{obj}^{(t_{i,j}-\delta)}$
(2) $o_j \notin l_{obj}^{(t_{i,j}-\delta)}$

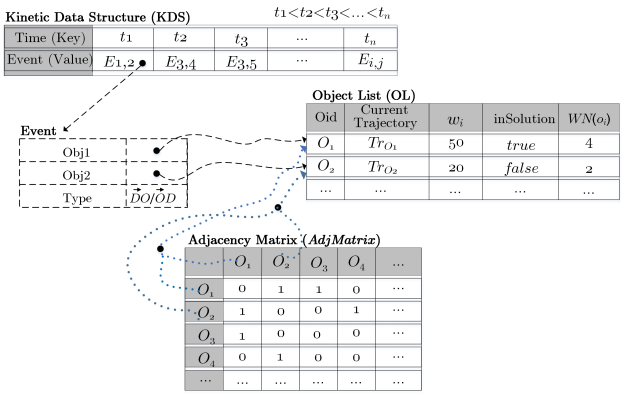
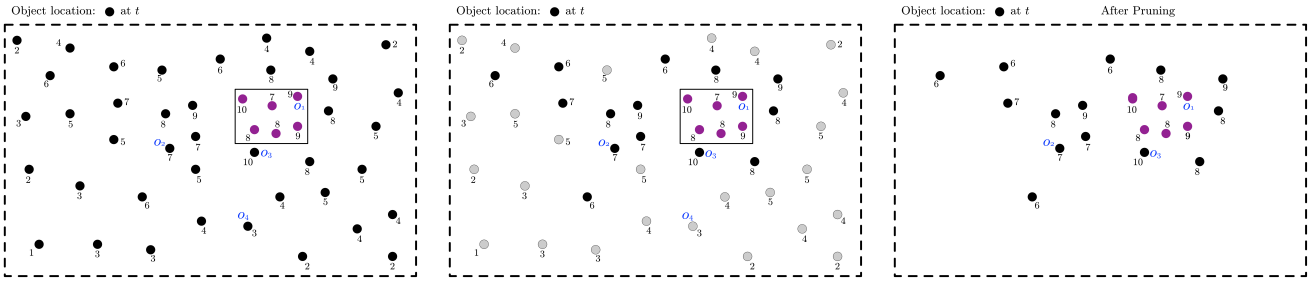


Figure 6: Data structures used.



(a) (b) (c)

Figure 7: An example showing the objects pruning scheme: (a) Objects' locations and $WN(o_i)$ values at t (b) Grey objects can be pruned using Lemma 3 in a $\vec{D}O$ event (c) Remaining objects after pruning at a $\vec{D}O$ event.

then $l_{obj}^{(t_{i,j}-\delta)}$ *remains a MaxRS solution after* $\vec{O}D_{ij}$ *(i.e, after* $t_{i,j}$*).*

To utilize Lemma 1 and 2, we maintain for each $o_i \in O_m$ the value of $WN(o_i)$, and whether or not the object is part of the current MaxRS solution. In Figure 6, two variables *inSolution* and $WN(o_i)$ are used for this purpose, updated accordingly during the processing of $\vec{D}O$ and $\vec{O}D$ events.

4.2 Objects Pruning

After filtering out many of the recomputations (Lemma 1 and Lemma 2), it is desirable to reduce the number of objects required in the recomputation. Towards that, we the following observations: (1) $WN(o_i) + w_i$ is an upper bound on possible MaxRS scores containing an object o_i; (2) $score_{max}$, the current MaxRS score, is a lower bound on possible MaxRS scores after a $\vec{D}O$ event; and (3) $score_{max} - min\{w_i, w_j\}$ is a lower bound on possible MaxRS scores after a qualifying $\vec{O}D_{ij}$ event. Let $E_{i,j}$ denote any event involving two objects o_i and o_j (be it $\vec{D}O_{ij}$ or $\vec{O}D_{ij}$). We have:

LEMMA 3. *After updating* $WN(o_i)$ *and* $WN(o_j)$ *at* E_{ij}*, an object* o_k *can be pruned before recomputing MaxRS if one of the following two conditions holds:*
(1) $E_{i,j}$ *is a* $\vec{D}O$ *event and* $WN(o_k) + w_k \leq score_{max}$
(2) $E_{i,j}$ *is an* $\vec{O}D$ *event and* $WN(o_k) + w_k \leq score_{max} - min\{w_i, w_j\}$

Example 1. Figure 7a demonstrates an example scenario with 46 objects. For the sake of simplicity, we only consider the counting variant (i.e., $\forall o_i \in O : w_i = 1$) in this example. The count of neighbors (i.e., $WN(o_i)$) for each object is shown as a label, and the current MaxRS solution is illustrated by a solid rectangle where $score_{max}$ (or, $count_{max}$) $= 6$. Members of l_{obj} are colored purple in Figure 7. Some of the objects are marked with an id (e.g., o_1, o_2, o_3, and o_4), so that they can be identified clearly in the text. In this scenario, to process any event, we will first update the appropriate $WN(o_i)$ and *inSolution* values. Then, suppose a new $\vec{D}O$ event is processed for one of the objects for which $WN(o_i) \leq 5$, e.g., between o_3 and o_4. Then that event will be pruned and MaxRS answer-set will remain the same as the maximum possible count of a MaxRS including that object will be $(5 + 1)$=6. Similarly, any $\vec{O}D$ event involving an object other than the purple ones would be filtered out. Figure 7b illustrates the application of Lemma 3, based on which all the objects in grey can be pruned during a $\vec{D}O$ event before recomputing MaxRS. Thus, after applying Lemma 3, we can prune 26 objects in linear time, i.e., going through the set of objects once and verifying the respective conditions. After pruning, 20 objects will remain (cf. Figure 7b) – only 43% of the total objects.

According to Lemma 1, a $\vec{D}O_{ij}$ event is not pruned when both $WN(o_i) + w_i > score_{max}$ and $WN(o_j) + w_j > score_{max}$ hold. Let us use $N(o_i)$ to denote the list of neighbors of any object o_i. Additionally, we employ $CN(o_i, o_j)$ to represent common neighbors of two objects o_i and o_j, i.e., $N(o_i) \cap N(o_j)$. In this setting, there are two possible cases:
Case 1: *Both* $o_i, o_j \notin l_{obj}$. The observation here is that if there exists a new MaxRS solution at a $\vec{D}O_{ij}$ event, then both o_i and o_j must be present in the new solution as only they are affected by the new $\vec{D}O$ event – all other objects (and their related attributes) remain the same. Additionally, for any MaxRS solution including both o_i and o_j, only the members of $CN(o_i, o_j)$ can be in l_{obj}.
Case 2: *Either* $o_i \in l_{obj}$ *or* $o_j \in l_{obj}$. Let us assume $o_i \in l_{obj}$. Then, if o_j overlaps with all objects $o_k \in l_{obj}$ (an $O(|l_{obj}|)$ check), then we can directly have a new MaxRS solution including o_j, i.e., $l_{obj} = l_{obj} \cup o_j$. If this check fails, we can follow the similar procedure as case 1. Note that, the case of both $o_i, o_j \in l_{obj}$ is not possible as it contradicts the concept of $\vec{D}O_{ij}$ event, i.e., o_i and o_j are mutually disjoint before $\vec{D}O_{ij}$. Based on the above observations, we have the following two lemmas:

LEMMA 4. *For an event* $\vec{D}O_{ij}$ *involving two objects* o_i *and* o_j*, we can prune all the objects except* o_i*,* o_j*, and* $CN(o_i, o_j)$ *before recomputing MaxRS.*

LEMMA 5. *For an event* $\vec{D}O_{ij}$ *involving two objects* o_i *and* o_j *where* $o_i \in l_{obj}$*, we can set* $l_{obj} \cup o_j$ *as the new MaxRS solution if* o_j *overlaps with all objects* $o_k \in l_{obj}$*.*

To take advantage of Lemma 4, we need to keep track of neighbors of all the objects in addition to $WN(o_i)$, which is the purpose of the adjacency matrix (*AdjMatrix* in Figure 6). We note that one could also maintain a list $N(o_i)$ for each object – however, although each approach would incur $O(n^2)$

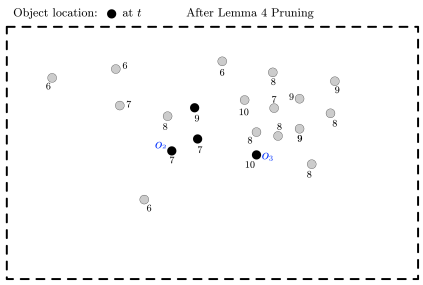


Figure 8: Application of Lemma 4 in $\vec{D}O$ events.

space overhead in the worst case, the adjacency matrix has certain advantages:

• Updating of the matrix information can be done in $O(1)$ time. For example, at a $\vec{D}O_{i,j}$ event we can directly set *AdjMatrix*[i][j]=1 and *AdjMatrix*[j][i]=1. Similarly, *AdjMatrix*[i][j] and *AdjMatrix*[j][i] can be set to 0 at an $\vec{O}D_{i,j}$ event.

• We can compute $CN(o_i, o_j)$ for two objects o_i and o_j efficiently by doing a *bit-wise AND* operation over *AdjMatrix*[i] and *AdjMatrix*[j].

Example 2. Suppose there is a new $\vec{D}O$ event between objects o_1 and o_3 in the example in Figure 7. The event will not be pruned because both $WN(o_1)$ and $WN(o_3) > 5$. As $o_1 \in l_{obj}$, we will first check if o_3 overlaps with all other members of l_{obj} (purple colored objects). As it does overlap with all the members of l_{obj}, we can directly output $l_{obj} \cup o_3$ as the new solution using Lemma 5. On the other hand, suppose the new $\vec{D}O$ occurs between o_2 and o_3. Using Lemma 4, we can prune all the objects except o_2, o_3, and $N(o_2) \cap N(o_3)$. This leaves us with only 4 remaining objects (cf. Figure 8) – 91.3% objects are pruned from the calculation. Obviously, score of the recomputed MaxRS will be less than the $score_{max}$ we already have (i.e., 6), and thus no change to the solution of Co-MaxRS will be made. We can see, Lemma 4 and Lemma 5 greatly optimizes processing of $\vec{D}O$ events.

4.3 KDS Properties and Algorithmic Details

Instead of a single line-segment, moving objects trajectories in practice are often polylines with vertices corresponding to actual location-samples. To cater to this, we introduce another kind of event, pertaining to an individual object – *line-change* event at a given time instant, denoted as $E_{lc}(o_i, t_{l_i})$. Suppose, for a given object o_i, we have $k+1$ time-samples during the period T as $t_{i1}, t_{i2}, \ldots, t_{i(k+1)}$, forming k line-segments. Note that the frequency of location updates may vary for different objects; even for a single object, the consecutive time-samples may have different time-gap. Initially, we insert the second time-samples for all the objects into the KDS as line-change events (cf. Figure 6). When processing $E_{lc}(o_i, t_{l_i})$ we need to compute: (a) Next $\vec{O}D$ events with the neighbors; and (b) Next $\vec{D}O$ events with other non-neighboring objects. We also need to insert a new line-change event at $t_{l_{(i+1)}}$ for o_i into the KDS. Thus, processing a line-change event takes $O(n)$ time. Note that a particular trajectory may start (appear) and/or finish its trip (disappear) at any time t, where $t_0 < t < t_{max}$ and we can use similar ideas to handle these special cases in $O(n)$ time.

KDS Properties: We proceed with briefly analyzing the properties of our proposed KDS-based structure (in the spirit of [2]), which shows that our adaption of KDS is responsive, efficient, local, and practically responsive.

(1) Number of certificates altered during an event (Responsiveness): Recall that we have two kinds of core events:

$\vec{D}O$ *Event:* At such an event we need to compute the time of the next $\vec{O}D$ event between the two objects and insert that to KDS if it falls within the given time-period T. Thus, only one new event (certificate) is added.

$\vec{O}D$ *Event:* For these events, we just need to process them, and no new event is inserted into KDS.

In both cases, the number is a small constant – conforming with the desideratum.

(2) The size of KDS (Compactness): In case of our adaptation of the KDS, we can have at most $O(n^2)$ $\vec{D}O$ and $\vec{O}D$ events at once. If we consider the additional line-change events for the polyline moving objects trajectories, there can be one such event for each object at any particular time, i.e., $O(n)$ such events. Thus, the size of KDS at a particular time is at most $O(n^2)$. However, as we will see in Section 5, in practice the size (total events) can be significantly smaller than this upper-bound – meeting the desideratum, i.e., $O(n^\epsilon)$ for some arbitrarily small $\epsilon > 0$.

(3) The ratio of internal and external events (Efficiency): In our KDS, the $\vec{D}O$ and $\vec{O}D$ events are external events (i.e., possibly causing changes to the Co-MaxRS answer-set), and the line-change events are internal. Thus, the ratio between total number of events and external events is $\frac{O(n^2)+O(n)}{O(n^2)}$, which is relatively small. This is a desired property of an *efficient* KDS [2].

(4) Number of certificates associated with an object (Locality): An object can have $n-1$ $\vec{D}O$ and $\vec{O}D$ events with the other objects, and 1 line-change event at a particular time instant, i.e., the number of events associated with an object is $O(n)$, which is an acceptable bound.

Subsequently, our adaption of KDS is responsive, efficient, local and, in practice, compact too.

Algorithmic Details: In Algorithm 2, we present the detailed method for maintaining Co-MaxRS for a given time period $[t_0, t_{max}]$. As mentioned, for each object, in addition to *WN* and *inSolution* variables, we also keep track of the active neighbors in RG via *AdjMatrix*. After initialization (line 1 and 2), the KDS is populated with all the initial events that fall within the given time-period (line 3) – a step taking $O(n^2)$ time. Then, we retrieve the current solution, i.e., the list of objects, and create a new time-interval of its validity, starting at t^{new}_{start} in lines 4-6. We update the *inSolution* values of related objects whenever we compute a new MaxRS solution, and discard an old one (lines 7, 15, and 16). Lines 8–19 process all the events in the KDS in order of their time-value, and maintain the Co-MaxRS answer-set throughout. The top event from the KDS is selected and processed using the function *EventProcess* (elaborated in Algorithm 3). After checking whether a new solution has been returned from *EventProcess*, the answer-set is adjusted in the sense of closing its interval of validity (t^{new}_{end})

Algorithm 2 Co-MaxRS (OL, R, t_0, t_{max})

1: $KDS \leftarrow$ An empty priority queue of events
2: $\mathbb{A}_{Co\text{-}MaxRS} \leftarrow$ An empty list of answers
3: Compute next event E_{next}, $\forall o_i \in OL$ and push to KDS
4: current $\leftarrow$ Snapshot of object locations at t_0
5: $(loc_{opt}, score_{max}, l_{obj}) \leftarrow$R_Location_MaxRS(current)
6: $t^{new}_{start} \leftarrow t_0$
7: Update *inSolution* variable for each o_i in l_{obj}
8: **while** KDS not EMPTY **do**
9: $E_{i,j} \leftarrow KDS.Pop()$
10: $(l'_{obj}, score_{max}) \leftarrow$ EventProcess($E_{i,j}, KDS, l_{obj}, score_{max}$)
11: **if** $l_{obj} \neq l'_{obj}$ **then**
12: $t^{new}_{end} \leftarrow t_i$
13: $\mathbb{A}_{Co\text{-}MaxRS}.Add(l_{obj}, [t^{new}_{start}, t^{new}_{end}))$
14: $t^{new}_{start} \leftarrow t_i$
15: Update *inSolution* variable for each o_i in l_{obj}
16: Update *inSolution* variable for each o_i in l'_{obj}
17: $l_{obj} \leftarrow l'_{obj}$
18: **end if**
19: **end while**
20: $t^{new}_{end} \leftarrow t_{max}$
21: $\mathbb{A}_{Co\text{-}MaxRS}.Add(l_{obj}, [t^{new}_{start}, t^{new}_{end}))$
22: **return** $\mathbb{A}_{Co\text{-}MaxRS}$

which, along with the corresponding l_{obj} are appended to $\mathbb{A}_{Co\text{-}MaxRS}(O_m, R, T)$ (for brevity, the ".Add()" notation is used). A modified version of the MaxRS algorithm from [22] is used where, in addition to the score, the list l_{obj} is also returned – cf. *R_Location_MaxRS* in line 5. Note that, the condition check at line 11 in implementation actually takes constant time, which we detect via setting a boolean variable during MaxRS computation.

The processing of a given KDS event $E_{i,j}$ is shown in Algorithm 3. In line 1, the *WN* of the relevant objects and *AdjMatrix* are updated. In lines 2–7, we compute new $\vec{O}D$ events and update the KDS. Lines 8–13 implement the ideas of Lemma 1 and Lemma 2, which takes $O(1)$ time. Lines 14–19 implement the idea of Lemma 5 to process a special kind of $\vec{D}O$ events. Line 20 introduces a new list OL', which will eventually retain only the unpruned objects. Lines 21-24 employ the idea of Lemma 4 for $\vec{D}O$ events. Lines 25–29 implement the ideas of objects pruning (Lemma 3), which takes $O(n)$ time. Finally, MaxRS is recomputed in lines 30–31 based on the current snapshot of the remaining moving objects in $O(n \log n)$ time (for brevity, we omitted handling line-change events in Algorithm 3). Lines 32–34 ensure that only valid computed values are returned, i.e., when $score'_{max} > score_{max}$ for $\vec{D}O$ events.

Discussion: In the worst-case, Co-MaxRS for n trajectories with k segments throughout the query time-interval, has $O(kn^2)$ events. In KDS, $O(n^2)$ events are added at the beginning, then at each of the $O(kn)$ line change events, $O(n)$ new events may be created, resulting in $O(kn^2)$ events in total. Observe that between two consecutive event-times t_{s-1} and t_s, there is a Co-MaxRS path of constant complexity (i.e., the centroid of R moves along a straight line-segment). As mentioned in Section 3, this follows from the fact that the Co-MaxRS solution covering a particular list l^s_{obj} in the sequence $(\mathbb{A}_{Co\text{-}MaxRS}(O_m, R, T))$ for the interval $[t_{s-1}, t_s]$, is the (maximum) intersection of sheared-boxes generated by the motion of the dual rectangles of the objects in l^s_{obj}. Thus,

Algorithm 3 EventProcess ($E_{i,j}$, KDS, l_{obj}, $score_{max}$)

1: Update $WN(o_i)$, $WN(o_j)$, and *AdjMatrix* accordingly
2: **if** $E_{i,j}.Type = \vec{D}O$ **then**
3: Compute E_{next} for objects o_i and o_j
4: **if** $E_{next} \neq$ NULL **and** $E_{next}.t \in [t_0, t_{max}]$ **then**
5: $KDS.Push(E_{next})$
6: **end if**
7: **end if**
8: **if** $E_{i,j}.Type = \vec{D}O$ **and** $(WN(o_i) + w_i \leq score_{max}$ **or** $WN(o_j) + w_j \leq score_{max})$ **then**
9: **return** $(l_{obj}, score_{max})$
10: **end if**
11: **if** $E_{i,j}.Type = \vec{O}D$ **and** $(o_i.inSolution = false$ **or** $o_j.inSolution = false)$ **then**
12: **return** $(l_{obj}, score_{max})$
13: **end if**
14: **if** $E_{i,j}.Type = \vec{D}O$ **and** Either $o_i/o_j \in l_{obj}$ **then**
15: $o_k \leftarrow o_j/o_i$
16: **if** o_k and l_{obj} are mutually overlapping **then**
17: **return** $(l_{obj} \cup o_k, score_{max} + w_k)$
18: **end if**
19: **end if**
20: $OL' \leftarrow OL$
21: **if** $E_{i,j}.Type = \vec{D}O$ **then**
22: $CN(o_i, o_j) \leftarrow$ Compute-CN ($AdjMatrix$, o_i, o_j)
23: $OL' \leftarrow CN(o_i, o_j) \cup \{o_i, o_j\}$
24: **end if**
25: **for all** o_k in OL' **do**
26: **if** $(E_{i,j}.Type = \vec{D}O$ **and** $WN(o_k) + w_k \leq score_{max})$ **or** $(E_{i,j}.Type = \vec{O}D$ **and** $WN(o_k) + w_k \leq score_{max} - min(w_i, w_j))$ **then**
27: Prune o_k
28: **end if**
29: **end for**
30: current $\leftarrow$ Snapshot of objects in OL' at t_i
31: $(loc'_{opt}, score'_{max}, l'_{obj}) \leftarrow$R_Location_MaxRS(current)
32: **if** $(E_{i,j}.Type = \vec{O}D)$ **or** $(E_{i,j}.Type = \vec{D}O$ **and** $score'_{max} > score_{max})$ **then**
33: **return** $(l'_{obj}, score'_{max})$
34: **end if**
35: **return** $(l_{obj}, score_{max})$

the worst-case combinatorial complexity of the path of the centroid of the Co-MaxRS solutions is $O(kn^2)$ – with a note that there may be discontinuities between consecutive locations of the centroids (i.e., the solution "jumps" from one location to another). The overall worst-case complexity when considering trajectories with multiple segments (i.e., polyline routes) is $O(kn^3 \log n)$.

We close this section with two notes:

(1) While the worst-case complexity of processing Co-MaxRS is high, such orders of magnitude are not uncommon for similar types of problems – i.e., detecting and maintaining flocks of trajectories [8]. However, as our experiments will demonstrate, the pruning strategies that we proposed can significantly reduce the running time.

(2) A typical query processing approach would involve *filtering* prior to applying pruning – for which an appropriate index is needed, especially when data resides on a secondary storage. Spatio-temporal indexing techniques abound since the late 1990s (extensions of R-tree or Quadtree variants,

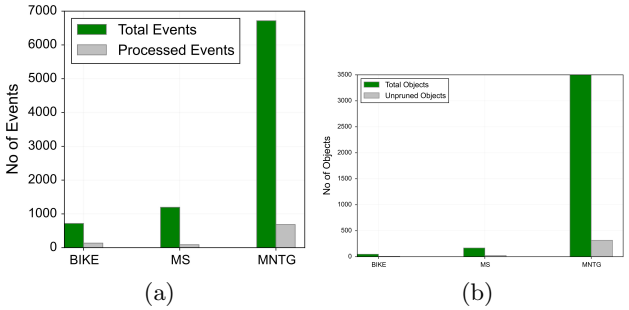
combined subdivisions in spatial and temporal domains, etc. [14, 19]). Throughout this work we focused on efficient in-memory pruning strategies, however, in Section 5 as part of our experimental observations, we provide a brief illustration about the benefits of using an existing index (TPR* tree [29]) for further improving the effects of the pruning. This, admittedly, is not a novel research or a contribution of this work, but it serves a two-fold purpose: (a) to demonstrate that our proposed approaches could further benefit by employing indexing; (b) to motivate further research for appropriate index structure.

5. EXPERIMENTAL OBSERVATIONS

Datasets: We used two real-world datasets and another synthetic one during our experiments. The first real-world dataset we used is the bicycle GPS (BIKE-dataset) collected by the researchers from University of Minnesota [10], containing 819 trajectories from 49 different participant bikers, and 128,083 GPS points. The second one is obtained from [34] (MS-dataset), which contains GPS-tracks from 182 users in a period of over five years collected by researchers at Microsoft with 17,621 trajectories in total, covering 1,292,951 km and over 50,176 hours (with GPS samples every 1-5 seconds). To demonstrate the scalability of our approach, we also used a large synthetic dataset (MNTG-dataset) generated using Minnesota Web-based Traffic Generator [18]. The generated MNTG-dataset consists of 5000 objects, and 50000 trajectories with 400 points each, where we set the option that objects are not constrained by the underlying network. For every object in the synthetic dataset, we generated its weight uniformly in the range from 1 to 50, while weights in Bike-dataset and MS-dataset (real-world datasets) were set to 1.

For each of the dataset used in the experiments, we considered one trajectory per object during a run and we averaged over all the runs to get representative-observations. The default values of the number of objects for BIKE, MS, and MNTG dataset are 49, 169, and 5000 respectively. The query time is set to the whole time-period (lifetime of trajectories) during a particular run for each respective dataset, and the base value of range area (R) for each of the BIKE, MS, and MNTG dataset is 500000, 100000, and 400000 m^2 respectively.

Implementations: We implemented all the algorithms in Python 2.7, aided by powerful libraries, e.g., Scipy, Matplotlib, Numpy, etc. We conducted all the experiments on a machine running OS X El Capitan, and equipped with Intel Core i7 Quad-Core 3.7 GHz CPU and 16GB memory. As no prior works exist that directly deal with the Co-MaxRS problem, we use Algorithm 1 as our baseline for comparison. In addition to the Algorithms 1, 2 and 3, we have two additional implementations[4]: (1) As mentioned at the end of Section 4.3, we added TPR*-tree index, to investigate the further benefits in terms of pruning with KDS; and (2) To demonstrate the benefits of our pruning schemes, we tested them against a trivial approximate-solution to Co-MaxRS: one that would periodically re-evaluate the query throughout its time-interval of interest. In other words, MaxRS is re-computed at each $t + \delta$, i.e., δ is a fixed time-period (default δ=5s).



(a) (b)

Figure 9: (a) Events Pruning (b) Objects Pruning.

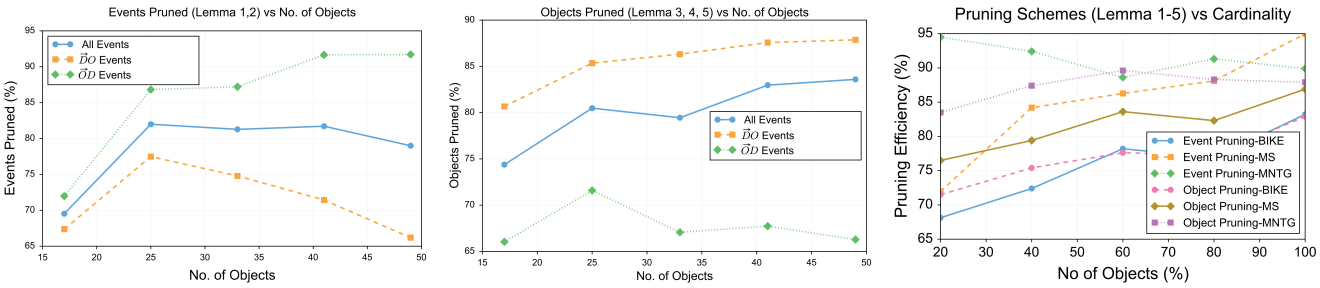
Performance of Pruning Strategies: Our first observations are shown in Figure 9a and they demonstrate the effectiveness of our events pruning strategy over both the real and synthetic datasets. The most amount of pruning is obtained in MS-dataset, while the other two datasets also show more than 80% pruning. Note that, the number of actual recomputation-events are well below the worst-case theoretical upper-bound, e.g., only 103 events are processed for 49 objects (trajectories) running for an hour in Bike-dataset. Similar results are obtained for the objects pruning scheme, as demonstrated in Figure 9b – indicating that the pruning schemes perform nearly equally well in all three datasets.

Impact of Cardinality: Figure 10 illustrates the impact of the cardinality on the effectiveness of our pruning methods. In Figure 10a, from the experiment done on the BIKE-dataset, we can deduce an interesting relation: as the datasize increases, more $\vec{O}D$ kind of events are pruned, whereas (cf. Figure 10b), objects pruning slightly decreases for $\vec{O}D$ as the datasize increases. On the other hand, $\vec{D}O$ events exhibit completely opposite behavior. This, in a sense, neutralizes the overall impact of the increase in cardinality for our pruning scheme. Figure 10c demonstrates the effect of increasing the cardinality of objects on the pruning schemes for all the dataset – hence, the label on the X-axis indicates the percentage of all the objects for the respective datasets.

Influence of Range Size: This experiment was designed to observe the effect of different range sizes, i.e., the area of $R - d_1 \times d_2$ over the pruning strategies. As shown in Figure 11a, increasing range area (the values on X-axis indicate multiples of the base-size for each dataset) results in fewer portion of events pruned. This occurs because as the area of R grows, there are more overlapping dual rectangles among the moving objects. Similarly, the growing rectangle size had adverse effects on the objects pruning scheme as well (cf. Figure 11b). We note, though, that even with quite large values of R (e.g., 50000 m^2) we have more than 60% of pruning through our proposed methods.

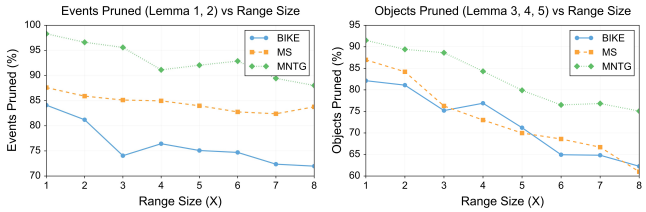
Benefits of indexing: Indexing the trajectories provides a filtering power which can be used as an additional pruning benefits (with respect to the Lemmas in Section 4) in terms of retrieving overlapping neighbors for any object. Figure 12 demonstrates benefits of indexing over varying cardinality (experiment done on MNTG Dataset). The running time using index is almost 100% times faster (half) for 5000 objects. We re-iterate that, as mentioned in Section 4.3, this is not a research contribution of the paper but only serves the purpose to demonstrate that an index is likely to yield

[4]We note that all the datasets and the source code of the implementation are publicly available at http://www.eecs.northwestern.edu/∼mmh683.



(a) (b) (c)

Figure 10: Impact of cardinality on the pruning schemes: (a) Different events pruning (BIKE-dataset) (b) Objects pruning (BIKE-dataset) (c) Overall objects and events pruning (all datasets).



(a) (b)

Figure 11: (a) Events pruning strategy; (b) Objects pruning strategy against varying range sizes.

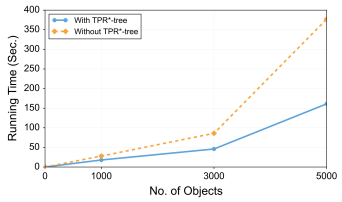


Figure 12: Potential impact of index.

further benefits for our proposed approaches.

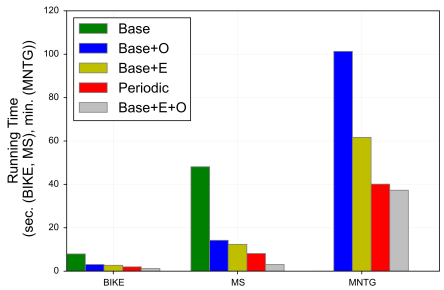
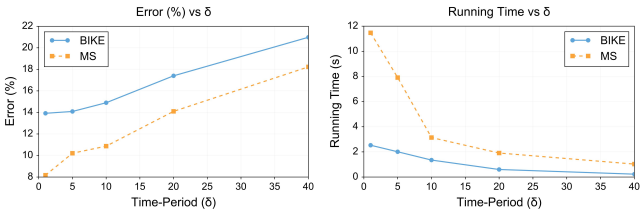


Figure 13: Running-time in different datasets.

Running Time Comparison: We ran the algorithms over the three datasets and the result is shown in Figure 13. This is the first experiment in which we also report observations regarding the periodical processing of the MaxRS – and it serves the purpose to provide a complementary illustration of the benefits of our methodologies. Namely, even if one is willing to accept an error in the result and perform only periodic snapshot MaxRS, our pruning techniques are still more efficient, while ensuring correct/complete answer set. The *Base*, *(Base+O)*, *(Base+E)*, *(Base+E+O)*, and *Periodic* in Figure 13 denote the base Co-MaxRS, base + objects pruning, base + events pruning, base + both events and objects pruning, and periodical processing of MaxRS (δ=5s), respectively. In case of MNTG-dataset, the average running time (for a set of trajectories) is shown in minutes, while for the other two datasets the unit it is shown in seconds. We omitted the average running time for the base algorithm over MNTG-dataset in Figure 13 which is more than 10 hours (to avoid skewing the graph). The base Co-MaxRS is the slowest among these algorithms, as it recomputes MaxRS at each event. The effect of both events and objects pruning schemes on running time is prominent, although events pruning exhibits a bigger impact individually (preventing unnecessary recomputations). When both pruning strategies are applied together, the algorithm speeds-up significantly – almost 6-15 times faster than the base algorithm over all the datasets – making it the fastest among all the evaluated algorithms.



(a) (b)

Figure 14: Impact of δ on (a) Error (b) Running Time of periodic-MaxRS.

Periodical Processing: The last observations illustrate the errors induced by periodical processing of MaxRS (periodic-MaxRS) to approximate Co-MaxRS. Note that we exclude performing periodic-MaxRS related experiments on the large synthetic dataset (MNTG-dataset) as the correctness, rather than scalability, is a concern. In Figure 14, the impact of (δ) is illustrated both on running time and correct-

ness. As δ increases the error in the approximation increases as well. Even for a small δ (e.g., $1s$), the respective error is still around 8-14% (cf. Figure 14a). Complementary to this, in Figure 14b, we see that as δ decreases, the running time increases too. For both Bike-dataset and MS-dataset, for small δ values ($\leq 5s$), average processing time is much longer than our proposed algorithm (*Base+E+O*) and yet it contains errors.

6. RELATED WORKS

The problem of MaxRS was first studied in the Computational Geometry community, with [11] proposing an in-memory algorithm to find a maximum clique of intersection graphs of rectangles in the plane. Subsequently, [22] used interval tree data structure to locate both (i) the maximum- and (ii) the minimum-point enclosing rectangle of a given dimension over a set of points. Although both works provide theoretically optimal bound, they are not suitable for large spatial databases, and a scalable external-memory algorithm – optimal in terms of the I/O complexity – was proposed in [5] (also addressing $(1-\epsilon)$-*approximate MaxRS* and *All-MaxRS* problems). More recently, the problem of indexing spatial objects for efficient MaxRS processing was addressed in [35]. In this work, we used the method of [22] to recompute MaxRS only at certain KDS events, however, we proposed pruning strategies to reduce the number of such invocations. We note that an indexing scheme based on a static sub-division of the 2D plane (cf. [5, 35]) need not to be a good approach for spatio-temporal data because the densities in the spatial partitions will vary over time, and we plan to investigate the problem of efficient indexing techniques for Co-MaxRS as part of our future work.

In [24], an algorithm to process MaxRS queries when the locations of the objects are bounded by an underlying road network is presented. Complementary to this, in [4] the solution is proposed for the *rotating-MaxRS* problem, i.e., allowing non axis-parallel rectangles. Recently, [1] proposed methods to monitor MaxRS queries in spatial data streams – objects appear or disappear dynamically, but do not change their locations. Although [1], [4], and [24] deal with interesting variants of the traditional MaxRS problem, they do not consider the settings of mobile objects.

In this work, we relied on the KDS framework, introduced and practically evaluated in [2]. The KDS-like data structure was used to process critical events at which the current MaxRS solution may change. To estimate the quality of a KDS, [2] considered performance measures such as the time-complexity of processing KDS events and computing certificate failure times, the size of KDS, and bounds on the maximum number of events associated with an object. We used the same measures to evaluate the quality of our approach.

Circular (Co-)MaxRS: A special note is in order for the, so called, circular MaxRS [3] – which is, the region R is a disk instead of a rectangle. Arguably, this problem is $\Theta(n^2)$ and one of the main reasons is that the combinatorial complexity of the boundary of the intersection of a set of disks is not constant (unlike axes-parallel rectangles). This, in turn, would increase the $n \log n$ factor in our algorithms to n^2 – and the continuous variant of the circular MaxRS implies maintaining intersections of sheared cylinders instead of sheared boxes. We also note that this case (counting variant) bears resemblance to works that have tackled problems in trajectory clustering [13]. More specifically, [8] introduced the concept of flocks as a group of trajectories who are moving together within a given disk and for a given time, and [12] introduced the (less constrained) concept of trajectory convoys. These works, while similar in spirit to a continuous variant of the circular MaxRS – have not explicitly addressed the problem of detecting (and maintaining) the disk which contains the maximum number of moving objects, nor have considered weights of the objects. We re-iterate that the results in [8] show that some of the proposed algorithms have complexities similar in magnitude to the worst-case complexity of the Co-MaxRS. An approximate solution to the static variant of the circular MaxRS was presented in [5] (approximating the disk with the minimum bounding square) and our current Co-MaxRS solution can be readily applied towards the approximated variant.

7. CONCLUSION AND FUTURE WORKS

We addressed the problem of determining the locations of a given axes-parallel rectangle R so that the maximum number of moving objects from a given set of trajectories is inside R. In contrast to the MaxRS problem first studied by the computational geometry community [11, 22], the Continuous MaxRS (Co-MaxRS) solution may change over time. To avoid checking the validity of the answer-set at every clock-tick, we identified the critical times at which the answer to Co-MaxRS may need to be re-evaluated, corresponding to – events occurring when the dual rectangles of the moving objects change their topological relationship. To speed up the processing of Co-MaxRS we used the kinetic data structures (KDS) paradigm and proposed two pruning heuristics: (1) eliminating events from KDS; and (2) eliminating the objects not affecting the answer (when re-computation of Co-MaxRS is necessary). While our algorithms mostly focused on the moving objects (resp. rectangles) defining the answer set, the possible volume(s) (in terms of 2D space + time) swept by the Co-MaxRS can be straightforwardly derived. Our experiments, over both real and synthetic data sets, showcased that the proposed heuristics enabled significant speed-ups in terms of the overall computation time from the upper bound on the time complexity.

There are numerous extensions of our work. One task is to devise a suitable indexing structure that will minimize the I/O overheads when trajectories data sets need to reside on a secondary storage or even on cloud [6], and to investigate the trade-offs between processing time vs. approximate answer to Co-MaxRS [5]. While, intuitively, our approaches seem "transferable" to the case of circular Co-MaxRS, we still need to have a more thorough investigation of the pruning effects in the KDS – and a related challenge is to investigate Co-MaxRS when the rectangles are in general positions (i.e., not restricted to be axes-parallel) [4]. In our solution there may be cases where Co-MaxRS has discontinuities – i.e., the current MaxRS needs to instantaneously change its location. Clearly, in practice one may want to have a realistic time-budget for the MaxRS to "travel" from one such location to another – which is another challenge to be addressed, in terms of lost precision. Other natural extensions of this setting are to investigate the k-variant of Co-MaxRS – i.e., the case of multiple mobile cameras jointly guaranteeing a continuous maximal coverage, as well as the effective management of Co-MaxRS for real time location updates.

8. REFERENCES

[1] D. Amagata and T. Hara. Monitoring MaxRS in spatial data streams. In *19th International Conference on Extending Database Technology, EDBT*, 2016.

[2] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1), 1999.

[3] B. M. Chazelle and D.-T. Lee. On a circle placement problem. *Computing*, 36(1-2), 1986.

[4] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, X. Cheng, and P. Chen. Rotating MaxRS queries. *Information Sciences*, 305, 2015.

[5] D. W. Choi, C. W. Chung, and Y. Tao. Maximizing Range Sum in external memory. *ACM Trans. Database Syst.*, 39(3):21:1–21:44, Oct. 2014.

[6] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce framework for spatial data. In *31st IEEE International Conference on Data Engineering, ICDE*, 2015.

[7] K. Feng, G. Cong, S. S. Bhowmick, W. C. Peng, and C. Miao. Towards best region search for data exploration. In *ACM SIGMOD International Conference on Management of Data*, pages 1055–1070. ACM, 2016.

[8] J. Gudmundsson and M. J. van Kreveld. Computing longest duration flocks in trajectory data. In *ACM GIS Conference*, 2006.

[9] R. H. Güting and M. Schneider. *Moving objects databases*. Elsevier, 2005.

[10] F. J. Harvey and K. J. Krizek. Commuter bicyclist behavior and facility disruption. Technical report, 2007.

[11] H. Imai and T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of algorithms*, 4(4), 1983.

[12] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1), 2008.

[13] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. 2005.

[14] S. Ke, J. Gong, S. Li, Q. Zhu, X. Liu, and Y. Zhang. A hybrid spatio-temporal data indexing method for trajectory databases. *Sensors*, 14(7), 2014.

[15] M. Koubarakis, T. Sellis, A. Frank, S. Grumbach, R. Güting, C. Jensen, N. Lorentzos, Y. Manolopoulos, E. Nardelli, B. Pernici, H.-J. Scheck, M. Scholl, B. Theodoulidis, and N. Tryfona, editors. *Spatio-Temporal Databases – the CHOROCHRONOS Approach*. Springer-Verlag, 2003.

[16] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity, 2011.

[17] M. Mas-ud Hussain, A. Wang, and G. Trajcevski. Co-MaxRS: Continuous Maximizing Range-Sum query. Technical Report NU-EECS-16-08, Dept. of EECS, Northwestern University, 2016.

[18] M. F. Mokbel, L. Alarabi, J. Bao, A. Eldawy, A. Magdy, M. Sarwat, E. Waytas, and S. Yackel. MNTG: An extensible web-based traffic generator. In *Advances in Spatial and Temporal Databases*. 2013.

[19] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-temporal access methods. *IEEE Data Eng. Bull.*, 26(2), 2003.

[20] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In *ACM SIGMOD*, 2004.

[21] Y. Nakayama, D. Amagata, and T. Hara. An efficient method for identifying MaxRS location in mobile ad hoc networks. In *Database and Expert Systems Applications - 27th International Conference, DEXA*, 2016.

[22] S. C. Nandy and B. B. Bhattacharya. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 29(8), 1995.

[23] N. Pelekis and Y. Theodoridis. *Mobility Data Management and Exploration*. Springer, 2014.

[24] T.-K. Phan, H. Jung, and U.-M. Kim. An efficient algorithm for Maximizing Range Sum queries in a road network. *The Scientific World Journal*, 2014, 2014.

[25] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient processing of top-k spatial preference queries. *Proceedings of the VLDB Endowment (PVLDB)*, 4(2), 2010.

[26] L. Sha, P. Lucey, Y. Yue, P. Carr, C. Rohlf, and I. A. Matthews. Chalkboarding: A new spatiotemporal query paradigm for sports play retrieval. In *21st International Conference on Intelligent User Interfaces, IUI*, 2016.

[27] S. Shekhar and S. Chawla. *Spatial databases: A tour*, volume 2003. Prentice Hall Upper Saddle River, NJ, 2003.

[28] Y. Tao, X. Hu, D. Choi, and C. Chung. Approximate MaxRS in spatial databases. *Proceedings of the VLDB Endowment (PVLDB)*, 6(13), 2013.

[29] Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In *International Conference on Very Large Data Bases (VLDB)*, 2003.

[30] C. R. Vicente, D. Freni, C. Bettini, and C. S. Jensen. Location-related privacy in geo-social networks. *IEEE Internet Computing*, 15(3), 2011.

[31] D. Wu, N. Mamoulis, and J. Shi. Clustering in geo-social networks. *IEEE Data Eng. Bull.*, 38(2), 2015.

[32] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *IEEE International Conference on Data Engineering (ICDE)*, 2005.

[33] Y. Zheng. Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3), 2015.

[34] Y. Zheng, L. Zhang, X. Xie, and W. Y. Ma. Mining interesting locations and travel sequences from GPS trajectories. In *ACM International Conference on World Wide Web*. ACM, 2009.

[35] X. Zhou, W. Wang, and J. Xu. General purpose index-based method for efficient MaxRS query. In *27th International Conference - DEXA*, 2016.