



NORTHWESTERN UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
Number: NU-EECS-16-08

May, 2016

Title

Co-MaxRS: Continuous Maximizing Range-Sum Query

Authors

Muhammed Mas-ud Hussain, Alex Wang, Goce Trajcevski

Abstract

We address the problem of efficient maintenance of Continuous Maximizing Range-Sum (Co-MaxRS) query for moving objects trajectories. The traditional MaxRS problem aims at finding a placement for a given (axes-parallel) rectangle R so that the number---or the sum of the weights---of objects (points) from a given set O in the interior of R is maximized. However, in the context of spatio-temporal data, due to the objects continuously changing their locations over time, the MaxRS solution for a particular time instant need not be a solution at another time instant. In this paper, we take a first step towards effective algorithmic solutions for this dynamic case. We devise the conditions under which a particular MaxRS solution may cease to be valid and propose efficient pruning strategies to speed-up the process of maintaining the correctness of the Co-MaxRS solution. We prove the correctness of our method and demonstrate via experiments, performed over both real and synthetic datasets, the benefits of the proposed methods in terms of efficiency and scalability in dealing with larger datasets.

Keywords

MaxRS, Kinetic Data Structures, Moving Objects Trajectories

Co-MaxRS: Continuous Maximizing Range-Sum Query

Muhammed Mas-ud Hussain, Alex Wang, Goce Trajcevski
Department of Electrical Engineering and Computer Science
Northwestern University
Evanston, Illinois 60208
{mmh683, alw231, goce}@eecs.northwestern.edu

ABSTRACT

We address the problem of efficient maintenance of Continuous Maximizing Range-Sum (Co-MaxRS) query for moving objects trajectories. The traditional MaxRS problem aims at finding a placement for a given (axes-parallel) rectangle R so that the number—or the sum of the weights—of objects (points) from a given set O in the interior of R is maximized. However, in the context of spatio-temporal data, due to the objects continuously changing their locations over time, the MaxRS solution for a particular time instant need not be a solution at another time instant. In this paper, we take a first step towards effective algorithmic solutions for this dynamic case. We devise the conditions under which a particular MaxRS solution may cease to be valid and propose efficient pruning strategies to speed-up the process of maintaining the correctness of the Co-MaxRS solution. We prove the correctness of our method and demonstrate via experiments, performed over both real and synthetic datasets, the benefits of the proposed methods in terms of efficiency and scalability in dealing with larger datasets.

1. INTRODUCTION

Moving Objects Databases (MOD) [8] are enabling technology for a wide range of applications that may demand some type of Location Based Services (LBS) [21] for mobile entities. Their applications include tracking in ecology and environmental monitoring, traffic management and online/targeted marketing, and military applications. The advances in sensing and communications technologies have generated large quantities of (*location, time*) data (O(Exabyte) [13]). Researchers in the spatio-temporal and MOD communities have focused on methods for efficient storage and retrieval of the whereabouts-in-time data, and on efficient approaches for processing various queries of interest, e.g., range, (k) nearest neighbor, alibi-queries, reverse nearest-neighbor, skyline, etc. Many of these queries have had their “predecessors” in traditional relational database settings, as well as in spatial databases [22] – however, due

to the movement of objects over time, their spatio-temporal variants became continuous (i.e., the answer-sets change over time) and even persistent (i.e., the answers not only change over time, but also depend on the history of the evolution/motion).

More recent research has addressed variants of the facility location problem and spatial preference queries in spatial databases [20, 26, 29] and, in particular, the efficient processing of the Maximizing Range Sum query (MaxRS) [5, 6]. Given a collection of weighted spatial points O and a rectangle R with fixed dimensions, MaxRS finds the location of R that maximizes the sum of the weights of the objects in its interior. Note that, contrary to the traditional range query, in the MaxRS setting one needs to determine the location where the sum of the weights (or the value of another objective function) is maximized. Originally, the MaxRS problem was tackled by the researchers from the field of computational geometry [11, 16] – however, motivated by its important LBS-applications (e.g., best location for a new franchise store with a limited delivery range, most attractive place for a tourist with a limited reachability range) more recent works have proposed scalable efficient solution for MaxRS in spatial databases [5, 6].

At the heart of the motivation for this work is the observation that in many applications involving mobile entities, a continuous variant of the MaxRS is paramount. Examples include: identifying the regions with highest density of tracked animals (e.g., gazelles) in different times; detecting ranges with densest traffic between noon and 6PM; (re)positioning of a drone-based camera to a location that maximizes its field of view coverage with respect to a set of mobile objects; etc. The Continuous MaxRS (Co-MaxRS) variant is also important in applications involving “interesting regions” for trajectories pattern mining, hotspots detection, etc. [25, 27]; many video games (e.g., World of Tanks) need to determine a position of maximal coverage in dynamic scenarios involving change of locations of players. The fundamental difference between MaxRS and Co-MaxRS is illustrated in Figures 1a and 1b. An instance of the MaxRS problem over a spatial database (assuming that the weights of all the objects are 1) is shown in Figure 1a, and the placement of the rectangle R indicated in solid line is the solution (count = 6). Other suboptimal solutions are shown in dashed lines. However, when objects are mobile, the placement of R at different time instants may need to be changed – as shown in Figure 1b for three different times (t_0 , t and t_{max}).

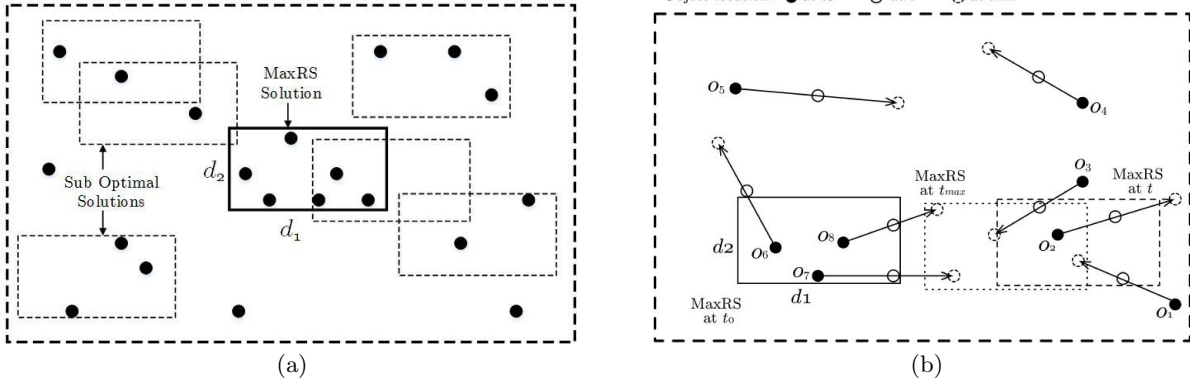


Figure 1: (a) An example of the MaxRS problem (b) An example of the Co-MaxRS problem, i.e., changing MaxRS solutions at different times for moving objects trajectories.

The main contributions of our work can be summarized as follows:

- We formally define the Co-MaxRS problem, and identify criteria (i.e., the critical times) under which a particular MaxRS solution may no longer be valid, and present algorithms for calculating such time instants and maintaining the correct Co-MaxRS solution.
- We propose two kinds of efficient pruning strategies to avoid expensive recomputation of Co-MaxRS solutions at certain critical times. The first strategy eliminates the recomputation altogether and the second strategy reduces the number of objects needed to be considered when recomputing the Co-MaxRS solution.
- We experimentally 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 times events and prune around 70% objects when recomputing Co-MaxRS.

The rest of this paper is organized as follows. Section 2 presents the basic technical background used in the rest of the paper. Section 3 formalizes the Co-MaxRS problem and the main data structures, and discusses the basic algorithm and the special case of *Static-MaxRS*, i.e., determining the static position which will maximize the number of objects passing through a rectangle over an entire time-interval. Section 4 presents our pruning strategies aiming at improved performance, along with the detailed algorithmic specifications. Section 5 presents the quantitative observations from our experiments. Section 6 positions the work with respect to the related literature, and Section 7 offers conclusions and directions for future work.

2. PRELIMINARIES AND PROBLEM FORMULATION

We now give an overview of the MaxRS problem and the existing solutions in static contexts, and introduce the concept of Kinetic Data Structures (KDS) used in the solution of Co-MaxRS.

2.1 MaxRS for Static Objects

Given a set of objects O and a query rectangle R , a MaxRS query finds a position of R within the specified space that

maximizes the sum of (the weights of) all the objects covered by R . Let $C(p, R)$ denote the region covered by R placed at a particular point p . We have:

Definition 1. (MaxRS) Given a set O of n points $O = \{o_1, o_2, \dots, o_n\}$, each o_i associated with a weight w_i and bounded within a rectangular area \mathbb{F} , the MaxRS query retrieves a position p within \mathbb{F} for an isothetic rectangle R of size $d_1 \times d_2$ such that $\sum_{\{o_i \in O \cap C(p, R)\}} w_i$ is maximal.

We define $\sum_{\{o_i \in O \cap C(p, R)\}} w_i$ as the *score* of R located at p . If $w_i = 1, \forall o_i \in O$, we refer the sum of weights as the *count*. An instance of the MaxRS problem (counting) is shown in Figure 1a. Note that there may be multiple solutions to the MaxRS problem and in case of ties, one is chosen randomly.

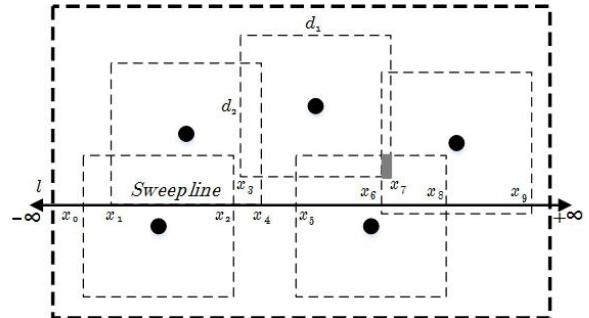


Figure 2: Transforming MaxRS problem into rectangle intersection problem.

To compute MaxRS for static objects, in-memory algorithms of $O(n \log n)$ time-complexity were proposed in [16]. More recently, a solution to the MaxRS problem in large-scale spatial databases, reducing the number of I/O's was presented in [6].

To illustrate the main idea, consider the counting variant of MaxRS where $\forall o_i \in O : w_i = 1$, and R has size $d_1 \times d_2$. An example of this is shown in Figure 2 where we have five objects (black-filled circles). This problem is transformed into a “dual” *rectangle intersection problem* (cf. [16]) as follows. We first draw a rectangle of size $d_1 \times d_2$ centered at each of the objects in O (see Figure 2). R covers o_i if and only if its

center is within the *dual rectangle* for o_i . Thus a rectangle covering the maximum number of points can be centered at any location within the maximum intersection of the dual rectangles (gray-filled area in Figure 2). The problem therefore becomes finding the area where the maximum number of rectangles intersect. Define a *rectangle graph* (RG), where vertices corresponds to the dual rectangles and an edge exists between two vertices if and only if the corresponding dual rectangles overlap. An area of maximum overlap in the dual representation corresponds to a maximum clique in RG.

Using the findings of [11] and the transformation described above, [16] provided an in-memory algorithm to solve the MaxRS problem in $O(n \log n)$ time and $O(n)$ space. Regarding the top and bottom edges of the rectangles as horizontal intervals, an *interval tree*—i.e., a binary tree on the intervals—was constructed in [16] and a horizontal line was swept in a bottom-up manner. 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. An example is shown in Figure 2. When the horizontal sweep-line is at position l , there are 9 intervals: $[-\infty, x_0]$, $[x_0, x_1]$, $[x_1, x_2]$, $[x_2, x_4]$, $[x_4, x_5]$, $[x_5, x_6]$, $[x_6, x_8]$, $[x_8, x_9]$, and $[x_9, +\infty]$ —with the count of the intervals being 0, 1, 2, 1, 0, 1, 2, 1, and 0 respectively. An interval with the maximum count during the whole sweeping process is returned as the final solution. Events in this algorithm are the top and bottom edges for the n rectangles. As there can be at most $2n$ events and each event takes $O(\log n)$ processing time, the whole algorithm takes $O(n \log n)$ time to complete.

2.2 Kinetic Data Structures

Kinetic data structures (KDS) are used to track attributes of interest in a geometric system. KDS is effective on systems where there is a set of values (e.g., location— x and y coordinate values) that are changing as a function of time in a known manner. KDS allows queries on a system at the (virtual) current time t , and additionally, two more operations: 1) advance the system to t ; and 2) alter the trajectory values to $f(t)$, e.g., location at time t . Initially, an instance of the data structure at t is stored (i.e., current values of the attributes of interest), which is augmented with a set of *certificates* proving its correctness at t . The failure times for each certificates are called *events*, which indicate that the data structure may no longer be an accurate representation of the state of the system. Thus, the next step is to compute the failure time of each certificate, a.k.a. events, and store the events in a priority queue sorted by their failure times. To advance to a future time $t + \delta$, we have to pop all the events having failure times $\leq t + \delta$ from the queue in-order, and fix the data structure so that it is accurate at that particular time and update the related certificates. In this paper, we utilize KDS to correctly 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. COMPUTING CO-MAXRS

In this section, we first discuss why a straightforward adaptation of the techniques used in MaxRS for spatial objects is not a viable option for Co-MaxRS and present a special case—Static-MaxRS, i.e., a static rectangle over moving objects. Subsequently, we address the Co-MaxRS problem.

Unless stated otherwise, for simplicity, the remainder of this paper will deal with the counting variants of the proposed problems. All of the techniques and algorithms presented can be adapted in a straightforward manner for the general nonnegative weight case.

Both [16] and [6] used interval trees as the underlying data structure of the planesweep algorithm. However, using only an interval tree to maintain MaxRS solutions continuously is inadequate because:

- (1) Interval tree needs to be built on the x -coordinate values of all the vertices of the dual rectangles. As the objects move, the interval tree has to be rebuilt at each time-instant, an $O(n \log n)$ operation.
- (2) We want to use the underlying data structure in an incremental manner in mobile settings. This is not possible in the case of interval trees as the plane-sweep procedure must sweep through all the top and bottom edges of the rectangles whenever there is a change in the interval tree.

3.1 Static-MaxRS Over Moving Objects

Instead of continuously tracking the MaxRS solutions over an entire time-period, one approach is to retrieve only the location within \mathbb{F} having the highest score or count during the whole time-period. We define Static-MaxRS as follows:

Definition 2. (Static-MaxRS) Given a set O_M of n moving objects $O_M = \{o_1, o_2, \dots, o_n\}$, where each $o_i = [(x_{i1}, y_{i1}, t_{i1}), \dots, (x_{i(k+1)}, y_{i(k+1)}, t_{i(k+1)})]$ has a trajectory within a rectangular area \mathbb{F} ; and a time-interval $T = [t_0, t_{max}]$, Static-MaxRS retrieves a location p of query rectangle R that covers the maximum number of objects over the entire time interval T , i.e., the number of objects intersecting the region $C(p, R)$ at least once is maximum throughout T .

For the example given in Figure 1b, the location of the MaxRS solution at t_{max} is a Static-MaxRS solution for the time-period $[t_0, t_{max}]$ as 5 objects— o_1, o_2, o_3, o_4, o_5 —are in that region (dotted rectangle in Figure 1b) at some time instant between $[t_0, t_{max}]$. If we have the Static-MaxRS solution for a given period T , we can return that location as an approximate answer for the Co-MaxRS problem for T , the intuition being that MaxRS solutions may be likely to revolve around *static hotspots* in real life data.

Assume for the time being, that for the given interval $T = [t_0, t_{max}]$, t_0 and t_{max} are consecutive sample time-points, i.e., each $o_i \in O$, moves along a single line-segment. For a query rectangle R of size $d_1 \times d_2$, let r_i be the rectangle centered at o_i , and S_i be the area swept by r_i over the given time-interval. S_i will be a rectangle if o_i is stationary or moves parallel to one of the axes, otherwise it will be a convex hexagon and can be computed in $O(1)$ time. We can derive an algorithm to compute Static-MaxRS combining the sweep-line approach of [16] and the subdivision overlay algorithm [3]. In our proposed sweep line algorithm, the event points will be the vertices of the S_i 's and the intersections of their edges. The status structure, denoted as ST , maintains the edges intersected by the sweep line and the count of the window to the right of each edge. The event

queue, denoted as Q , maintains the set of all the vertices of S_i and intersections above the sweep line of line-segments currently adjacent in ST in order from bottom to top.

Analysis and Discussion: We can handle each event point in $O(\log n)$ time by using balanced binary search trees for ST and Q . Thus the algorithm runs in $O((n + I) \log n)$ time where I is the number of intersection, and $I = O(n^2)$. Static-MaxRS has its own applications, e.g., finding the area where maximum number of vehicles cross throughout an entire day. However, it has no notion of duration, i.e., the algorithm is blind to how long each point is inside R . Also, it may not be a very good approximation of the Co-MaxRS problem, as MaxRS solutions at two different time-instants can vary significantly. Recall the example in Figure 1b, Static-MaxRS returns the dotted rectangle as the query answer for the time-period $[t_0, t_{max}]$. But, this is the MaxRS solution only at time t_{max} . At t_0 and t , the actual MaxRS solution has counts of 3, whereas the count of the Static-MaxRS solution is only 1— o_2 at t_0 and o_3 at t (cf. Figure 1b).

3.2 Computing Co-MaxRS: Basic Approach

Formally, the Continuous MaxRS (Co-MaxRS), is defined as follows:

Definition 3. (Co-MaxRS) Given a set O_M of n moving points $O_M = \{o_1, o_2, \dots, o_n\}$, where each $o_i = [(x_{i1}, y_{i1}, t_{i1}), \dots, (x_{i(k+1)}, y_{i(k+1)}, t_{i(k+1)})]$ is associated with a trajectory within a rectangular area \mathbb{F} ; and a time-interval $T = [t_0, t_{max}]$, Co-MaxRS returns a sequence of MaxRS solutions (i.e., a list of collections of objects) for a query rectangle R throughout the time-period T . Specifically, the Co-MaxRS answer-set contains a time-ordered sequence of pairs (l_{obj}, t_{range}) , where for any time instant $t_i \in t_{range} \subseteq T$, the objects in l_{obj} determine possible location(s) for R that is a MaxRS at t_i .

We note that we discuss the entire volume swept by the instantaneous MaxRS's throughout each t_{range} in Section 4.3.

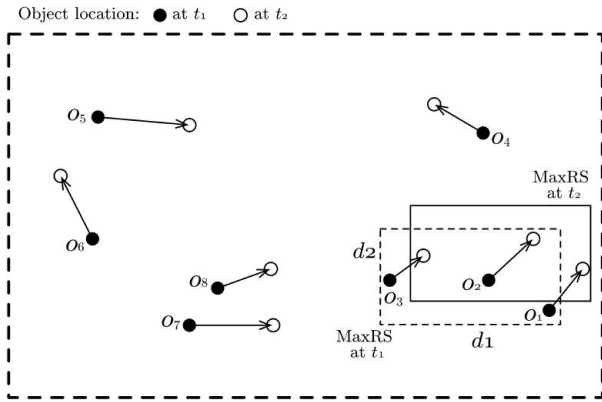


Figure 3: Optimal location of R changes from t_1 to t_2 , although the objects in the solution (O_1, O_2, O_3) are the same.

Instead of maintaining a centroid-location (equivalently, a region) as a 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) even for small object movements, the optimal location of the query rectangle can change while objects participating in the MaxRS solution stay the same; (2) the set of the objects that form the MaxRS solution can only change at certain time instants and, thus, can be tracked more efficiently. An example is shown in Figure 3. 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 are within the MaxRS solution, the optimal location itself has shifted due to the movement of the objects. Suppose, there are m objects in the l_{obj} list at a particular time instant. Using the ideas described in Section 2, we only need to find the intersection of m rectangles to retrieve the optimal MaxRS location. Thus, both the location and count/score of MaxRS solutions can be retrieved from l_{obj} when needed in $O(m)$ time. For the example given in Figure 1b, Co-MaxRS answer-set is: $\{((o_6, o_7, o_8), [t_0, t - 1]), ((o_1, o_2, o_3), [t, t_{max} - 1]), ((o_1, o_3, o_7, o_8), [t_{max}, t_{max}])\}$.

We now describe our proposed solution for the Co-MaxRS problem, aiming at identifying when recomputation of the MaxRS may be needed due to the possibility of a change in the solution. We consider the dual rectangle intersection problem under the mobile settings and consequently, keep track of the area of maximum overlap given moving rectangles.

Consider the example in Figure 4, with 10 objects: $\{o_1, o_2, \dots, o_{10}\}$. Let r_i denote the dual rectangle for an object o_i . Assume that only o_2 and o_5 are moving: o_2 is moves west, and o_5 is moves north (orange rectangles and arrows). Figure 4a, shows the locations of objects at t_1 and the MaxRS solution is comprised of o_1, o_2, o_3 , and o_4 (blue colored objects in Figure 4a). In this setting, r_2 and r_5 do not overlap. Figure 4b 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 4c 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 previous solution. We observe that the MaxRS 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 only when two overlapping rectangles became disjoint. Thus, the solution of Co-MaxRS changes only when two rectangles change their topological relationship from *disjoint* to *overlapping* ($\bar{D}O$), or from *overlapping* to *disjoint* ($\bar{O}D$). Note that 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.

3.2.1 Kinetic Maintenance of Co-MaxRS

To maintain the correct Co-MaxRS solution over time, we use the *Kinetic Data Structure* (KDS) paradigm. Recall that when an object/rectangle moves there are two kinds of changes:

(1) *Continuous Deformation:* As the location of the moving rectangles change, the region of maximum overlap may change but the set of objects constituting the Co-MaxRS solution stay the same.

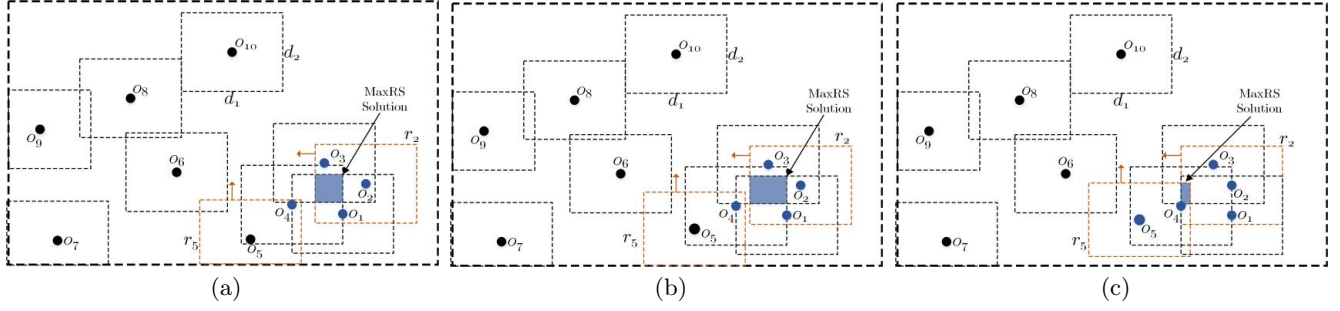


Figure 4: Co-MaxRS answer can only change when two rectangles' relationship changes from overlap to disjoint (or, vice-versa). Object locations at: (a) t_1 (b) t_2 (c) t_3 .

(2) *Topological Change*: Due to the movement of the rectangles, a \vec{DO} or \vec{OD} transition occurs for a pair of rectangles.

Thus, the Co-MaxRS answer-set can be kinetically maintained by tracking only the topological changes using the KDS framework, with a set of certificates (overlap/disjoint) proving the correctness of the Co-MaxRS solution at a particular time instant. Recall that, in KDS parlance, events (i.e., the failure times for the certificates) indicate a topological change, and are maintained in a priority queue in the order of their time of occurrence.

In the case of Co-MaxRS, the only two kinds of events that can cause a topological change are (1) \vec{DO} and (2) \vec{OD} , each associated with a pair of moving rectangles. Assuming a straight-line motion, in $[t_0, t_{max}]$, there could be at most one \vec{DO} and one \vec{OD} event between each pair of moving rectangles—shown in Figure 5. Initially at t_0 , rectangle r_1 (blue-colored) and r_2 (orange-colored) do not overlap. As they move along the straight-line, at time t_1 , they begin overlapping, i.e., \vec{DO} . Moving further along the line, near time t_2 , r_1 and r_2 become disjoint again, i.e., \vec{OD} . A pair of moving rectangles may also have one or zero \vec{DO} or \vec{OD} events. Given the moving objects trajectories, we can compute the times of occurrence of \vec{DO} (resp. \vec{OD}) in constant time.

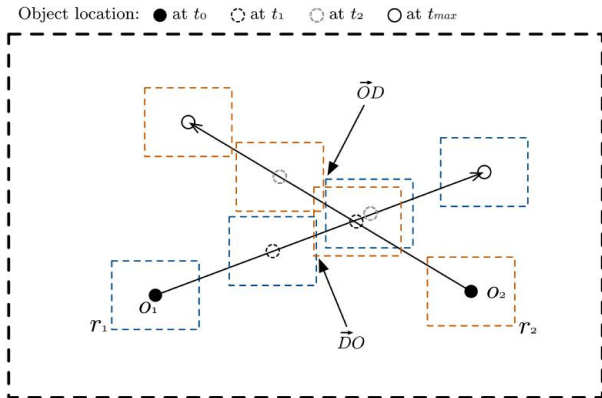


Figure 5: Possible events between two different moving rectangles: \vec{DO} and \vec{OD} .

3.2.2 Data Structures and Basic Algorithm

Figure 6 depicts the underlying data structures used to maintain Co-MaxRS answer-set using KDS.

Object List (OL): A list for each object $o_i \in O$, stores its current trajectory Tr_{o_i} (i.e., snapshots of location at t_0 and t_{max}), its number of neighbors $N(o_i)$ in the rectangle graph, and whether or not the object is part of the MaxRS solution.

Kinetic Data Structures (KDS): Figure 6 illustrates the underlying KDS (event queue), and its relation with the OL. Each event E_i is associated with a time t_i , where $t_0 < t_i < t_{max}$. KDS maintains an event queue, where the events are sorted according to the time, i.e., $t_1 < t_2 < t_3 < \dots < t_n$. An event E_i has pointers to its related objects—two object ids, and the type of the event—(\vec{DO} or \vec{OD}).

Suppose we have n objects in OL. Initially at t_0 , we can determine the time of \vec{DO} and/or \vec{OD} events between a pair of rectangles in constant time. If an event time falls within the time-period T , we insert that event into the KDS event queue. There are $O(n^2)$ pairs of rectangles, thus this step would take $O(n^2)$ time and produces at most $O(n^2)$ event points. Next, we process the events from the KDS in order of their occurrence time. At each event E_i , we recompute the current MaxRS on the snapshot of the moving objects database at t_i ; compare with the previous MaxRS solution to check if we have a change or not; and update the Co-MaxRS answer-set accordingly. The recomputation is bounded by $O(n \log n)$ time. Thus, the total time-complexity of our proposed algorithm to maintain Co-MaxRS is $O(n^3 \log n)$ for single line trajectories.

4. EFFICIENT MAINTENANCE OF CO-MAXRS

We now present the details of our approach to efficiently process Co-MaxRS. First, we discuss two strategies which, although do not improve the worst-case time complexity of the Co-MaxRS computation, may significantly reduce computational overheads. We aim to: (1) reduce the number of recomputations of a MaxRS; and (2) reduce the total number of objects required when recomputing a MaxRS solution. Subsequently, we present and analyze the corresponding algorithms.

4.1 Pruning KDS Events

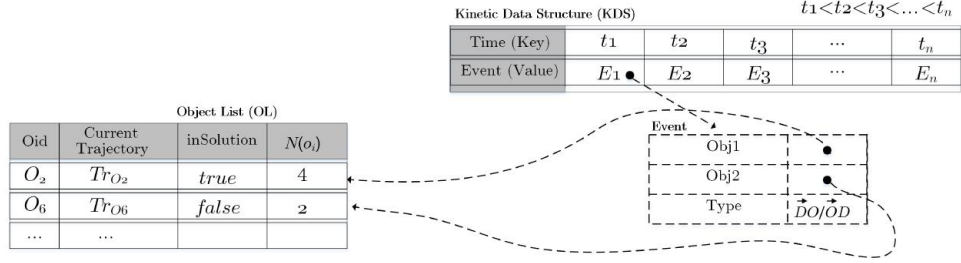


Figure 6: Data structures used in maintaining Co-MaxRS.

The MaxRS problem amounts to retrieving the maximum clique in the rectangle graph RG —i.e., a complete subgraph $RG' \subset RG$. In Section 3, we have shown that when the objects are moving, the rectangles around them can alternate between overlap/disjoint, which means that edges are dynamically added and removed to RG . One intuition then is to use the procedures of dynamic clique maintenance proposed in [23] and [18] incrementally to improve the efficiency of the Co-MaxRS base algorithm. These works propose efficient incremental algorithms to maintain the list of all maximal cliques in dynamic graphs. This is not suitable in our context for two reasons: (1) [16] proved that there could be $O(n^2)$ maximal cliques in RG . Maintaining a list of all maximal cliques would therefore amount to a space overhead of $O(n^2)$; (2) [16, 18] have not proposed a suitable extension to dynamically track a maximum clique ([23] proposes a method to dynamically track a multiple clique given either edge insertions or deletions but not both). In our settings, dynamic additions and removals of edges in RG happen at events in KDS, and a naive approach would compute the maximum clique at a given time instant from the list of all maximal cliques—however, the time complexity for that is $O(n^2)$. The following properties allow us to filter out \vec{DO} and \vec{OD} events where recomputing the MaxRS is unnecessary.

\vec{DO} : Call o_j a *neighbor* of o_i at a time t , if they are neighbors in RG . Let $N(o_i)$ denote the current number of neighbors of an object o_i , and let $count_{max}$ denote the size of the maximum clique. Note that $N(o_i)$ is exactly the number of dual rectangles that intersect the dual rectangle of o_i and $count_{max}$ is the maximum overlap.

LEMMA 1. Consider the event \vec{DO}_{ij} for two objects o_i and o_j . Let l_{obj} be a MaxRS solution before \vec{DO}_{ij} . If one of the following two inequalities holds:

- (1) $N(o_i) + 1 \leq count_{max}$
- (2) $N(o_j) + 1 \leq count_{max}$

then l_{obj} remains a MaxRS solution after \vec{DO}_{ij} .

PROOF. An intersecting event is equivalent to adding an edge in RG . A new clique after \vec{DO}_{ij} in RG must contain o_i and o_j . Without loss of generality, assume that $N(o_i) + 1 \leq count_{max}$. The upper bound on the size of the maximum possible clique containing o_i is $N(o_i) + 1$ (including itself). Thus, if the size $count_{max}$ of l_{obj} is at least $N(o_i) + 1$, then we can choose l_{obj} as the MaxRS solution, as in the case of a tie we can choose any solution arbitrarily. \square

\vec{OD} : In case of \vec{OD} , the intuition is much simpler—the count of a MaxRS solution can only decrease. Thus, we have:

LEMMA 2. Consider the event \vec{OD}_{ij} for two objects o_i and o_j . Let l_{obj} be a MaxRS solution before \vec{OD}_{ij} . If one of the following two conditions holds:

- (1) $o_i \notin l_{obj}$
- (2) $o_j \notin l_{obj}$

then l_{obj} remains a MaxRS solution after \vec{OD}_{ij} .

PROOF. We again use the ideas of RG and maximum cliques to prove this lemma. An \vec{OD}_{ij} event amounts to removing an edge in RG . Without loss of generality, assume that $o_i \notin l_{obj}$. By assumption, l_{obj} is a maximum clique and thus a maximal clique before \vec{OD}_{ij} . l_{obj} remains a maximal clique after \vec{OD}_{ij} as it is still connected and no new vertex can become connected to all vertices in l_{obj} . For the sake of contradiction, assume that some l'_{obj} after \vec{OD}_{ij} has size strictly greater than $|l_{obj}|$. As l'_{obj} must have been a clique before \vec{OD}_{ij} , this contradicts the optimality of l_{obj} before \vec{OD}_{ij} . It follows that l_{obj} is a MaxRS solution after the \vec{OD}_{ij} event. \square

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

4.2 Objects Pruning

Although we can filter-out many of the recomputations using Lemma 1 and 2, we may have to recompute the MaxRS solution at certain qualifying time instants. In such cases, it is desirable to reduce the number of objects considered in the recomputation. Using similar reasoning as in Section 4.1, we can prune objects based on the following observations: (1) $N(o_i) + 1$ is an upper bound on possible MaxRS counts containing o_i ; (2) $count_{max}$, the current MaxRS count, is a lower bound on possible MaxRS counts after a \vec{DO} event; and (3) $count_{max} - 1$ is a lower bound on possible MaxRS counts after a \vec{OD} event. Thus, we have:

LEMMA 3. For an event $E_{i,j}$ involving two objects o_i and o_j , an object o_k can be pruned before recomputing MaxRS if one of the following two conditions holds:

- (1) E_i is a \vec{DO} event and $N(o_k) + 1 \leq count_{max}$
- (2) E_i is a \vec{OD} event and $N(o_k) + 2 \leq count_{max}$

PROOF. For an object o_k , the upper bound of the maximum possible clique size including itself is $N(o_k) + 1$. For a \vec{DO} event, the original maximum clique remains complete. Thus, the size of the original maximum clique is a lower

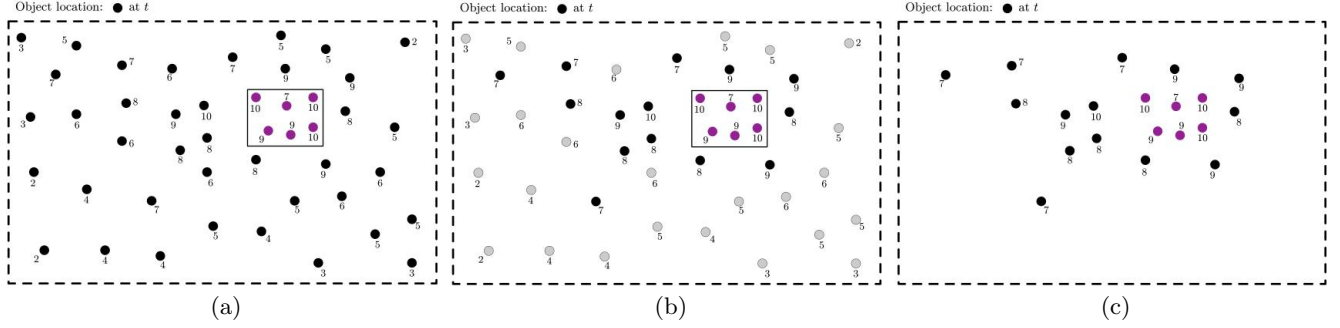


Figure 7: An example showing the objects pruning scheme: (a) Objects locations and $N(o_i)$ values at t (b) Grey objects can be pruned using Lemma 3 in a $\bar{D}O$ event (c) Remaining objects after pruning at a $\bar{D}O$ event.

bound for the new maximum clique. Thus, we can prune all the objects o_k for which $N(o_k) + 1 \leq count_{max}$. For an $\bar{O}D$ event, the size of the original maximum clique can decrease by at most one; this is lower bound on the size of the new maximum clique. Thus, we can prune all the objects o_k for which $N(o_k) + 2 \leq count_{max}$. \square

Example 1. Figure 7a demonstrates an example scenario with 46 objects. The number of neighbors (i.e., $N(o_i)$) for each object is shown as a label, and the current MaxRS solution is illustrated by a solid rectangle where $count_{max} = 6$. Members of l_{obj} are colored purple in Figure 7. In this scenario, suppose a new $\bar{D}O$ event is processed for one of the objects for which $N(o_i) \leq 6$. Then that event will be pruned after updating the appropriate $N(o_i)$ and $inSolution$ values. Similarly, any $\bar{O}D$ 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 $\bar{D}O$ 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 checking the respective conditions. After pruning, 20 objects will remain (cf. Figure 7c)—only 43% of the total objects.

4.3 KDS Properties and Algorithmic Details

Instead of a single line-segment, moving objects trajectories are typically poly-lines with vertices at actual location-samples. To handle this, we introduce another kind of event, pertaining to an individual object—*line-change* event at a given time instant, denoted as $E_{lc}(o_i, t_i)$. Suppose, for a given object o_i , we have $k+1$ time-samples during the period T as $t_{i1}, t_{i2}, \dots, t_{i(k+1)}$, 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 $(E_{lc}(o_i, t_i))$ we need to compute: (a) next $\bar{O}D$ events with the neighbors; and (b) next $\bar{D}O$ events with other non-neighboring objects. Additionally, we insert a new line-change event at $t_{i(i+1)}$ for o_i into the KDS. Thus, processing a line-change event takes $O(n)$ time, and we can use similar ideas to even handle newly appearing and/or disappearing objects. The worst-case time-complexity of our proposed algorithm in case of poly-line trajectories is $O(kn^3 \log n)$.

We now proceed with briefly analyzing the properties of our proposed KDS-like structure (cf. [1, 2]).

(1) Number of certificates altered during an event: This property of a KDS is called “Responsiveness”. Recall that we have two kinds of core events:

$\bar{D}O$ Event: At such an event we need to compute the time of the next $\bar{O}D$ 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.

$\bar{O}D$ Event: For these events, we just need to process them, and no new certificate is inserted into KDS. In both cases, the number is constant—conforming with the desideratum.

(2) The size of KDS: In case of our adaptation of the KDS, we can have at most $O(n^2)$ $\bar{D}O$ and $\bar{O}D$ events at once. If we consider the additional line-change event for the poly-line moving object 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.

(3) The ratio of internal and external events: In our KDS, the $\bar{D}O$ and $\bar{O}D$ events are external events (i.e., causing topological changes), and the line-change events are external. According to the discussion above, 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 KDS [2].

(4) Number of certificates associated with an object: The number of events associated with an object is $O(n)$, as an object can have $n-1$ $\bar{D}O$ and $\bar{O}D$ events with the other objects, and 1 line-change event at a particular time instant.

4.3.1 Processing Algorithms

In Algorithm 1, we present the detailed method for maintaining Co-MaxRS for a given time period $[t_0, t_{max}]$. For each object, we keep track of: the number of its neighbors; whether an object is in the current MaxRS solution or not; and the list of current neighbors (needed for line-change and disappearing event). After initialization (line 1 and 2), the KDS is populated with all the 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-range t_{range} in lines 4-6. We update the $inSolution$ values of related objects whenever we compute

Algorithm 1 Co-MaxRS (OL, t_0, t_{max})

```
1:  $KDS \leftarrow$  An empty priority queue of events
2:  $AnswerSet \leftarrow$  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:  $(loc_{opt}, count_{max}, l_{obj}) \leftarrow$  Modified-MaxRS ( $current$ )
6:  $t_{range}.begin \leftarrow t_0$ 
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:   $(l'_{obj}, count_{max}) \leftarrow EventProcess(E_{i,j}, KDS, l_{obj},$   
    $count_{max})$ 
11:  if  $l_{obj} \neq l'_{obj}$  then
12:     $t_{range}.end \leftarrow t_i$ 
13:     $AnswerSet.Add(l_{obj}, t_{range})$ 
14:     $t_{range}.begin \leftarrow t_i$ 
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:  $t_{range}.end \leftarrow t_{max}$ 
21:  $AnswerSet.Add(l_{obj}, t_{range})$ 
22: return  $AnswerSet$ 
```

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, and maintain the Co-MaxRS answer-set throughout. The top event from the KDS is selected and processed using the function *EventProcess* (cf. Algorithm 2). After checking whether a new solution has been returned from *EventProcess*, the answer-set, t_{range} , and l_{obj} are adjusted accordingly. A modified version of the MaxRS algorithm from [16] is used where, in addition to the count, the list of objects that are covered by the optimal location of R are also returned. 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 2. In line 1, the $N(o_i)$ value and list of neighbors of the relevant objects are updated. Lines 2-7, compute new $\vec{O}D$ events and update the KDS. Lines 8 - 13 implement the ideas of Lemma 1 and Lemma 2. Lines 14 - 17 implement the ideas of objects pruning (Lemma 3), which takes $O(n)$ time. Finally, MaxRS is recomputed in lines 20 - 21 based on the current snapshot of the remaining moving objects. Note that, we omitted handling line-change events in Algorithm 2 for brevity.

Discussion: To return to the spatio-temporal domain, we may want to construct a Co-MaxRS path — a path for the center of R such that at each time instant, R is a MaxRS. This path will inherently be disjoint. We show that there exists a Co-MaxRS path of constant combinatorial complexity.

In the worst-case, Co-MaxRS for n trajectories with k segments each throughout the query time-interval has $O(kn^2)$ events. This can be seen as $O(n^2)$ events are added at the beginning, then at each of the $O(kn)$ line change events, $O(n)$ new events may be created. This results in a total of $O(kn^2)$ events.

Algorithm 2 EventProcess ($E_{i,j}, KDS, l_{obj}, count_{max}$)

```
1: Update  $N(o_i)$  and neighbors of  $o_i$  and  $o_j$  accordingly
2: if  $E_{i,j}.Type = \vec{D}O$  then
3:   Compute  $E_{next}$  for objects  $o_i$  and  $o_j$ 
4:   if  $E_{next} \neq \text{NULL}$  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  $(N(o_i) + 1 \leq count_{max}$  or  
    $N(o_j) + 1 \leq count_{max})$  then
9:   return  $(l_{obj}, count_{max})$ 
10: end if
11: if  $E_{i,j}.Type = \vec{O}D$  and  $(o_i.inSolution = \text{false}$  or  
    $o_j.inSolution = \text{false})$  then
12:   return  $(l_{obj}, count_{max})$ 
13: end if
14: for all  $o_i$  in  $OL$  do
15:   if  $(E_{i,j}.Type = \vec{D}O$  and  $N(o_i) + 1 \leq count_{max})$  or  
      $(E_{i,j}.Type = \vec{O}D$  and  $N(o_i) + 2 \leq count_{max})$  then
16:     Prune  $o_i$ 
17:   end if
18: end for
19:  $current \leftarrow$  Snapshot of unpruned object locations at  $t_i$ 
20:  $(loc_{opt}, count_{max}, l_{obj}) \leftarrow$  Modified-MaxRS( $current$ )
21: return  $(l_{obj}, count_{max})$ 
```

Between consecutive event points t_1, t_2 , there is a Co-MaxRS path of constant complexity. We see this as follows. In the dual interpretation, a Co-MaxRS solution l_{obj} for the interval $[t_1, t_2]$ is a maximum intersection of sheared-boxes (the dual rectangles moving through time). As the intersection of convex solids is convex, the intersection of the dual sheared-boxes of l_{obj} is convex. A path in this convex solid is a Co-MaxRS path. It follows that for any position of R that covers l_{obj} at time t_1 and any position of R that covers l_{obj} at time t_2 , R can cover l_{obj} over the entire time interval by moving along the path between the two positions at constant speed, and there is a path of constant complexity. Thus, there exists a Co-MaxRS path with combinatorial complexity $O(kn^2)$.

We close this section with a note that a typical query processing would involve filtering prior to applying pruning—for which an appropriate index is needed. Spatio-temporal indexing techniques abound, since the late 1900s, e.g., extensions of R-tree or Quadtree variants, combined subdivisions in spatial and temporal domains, etc. [15, 17, 24]. A particular structure that may be applicable to Co-MaxRS settings is HBSTR-tree—a hybrid index structure consisting of spatio-temporal R-tree, B*-tree and Hash table [12]—however, throughout this work we focus on efficient pruning strategies and the incorporation of access methods for secondary storage is a subject of the future work.

5. EXPERIMENTAL OBSERVATIONS

We now present the detailed setup of our experiments and the results of evaluating the proposed approaches.

5.1 Setup

Datasets: We used both real-world and synthetic datasets during our experiments. The first dataset we used is a real-world bicycle GPS dataset (BIKE-dataset) collected by the

researchers from University of Minnesota [9], containing 819 trajectories from 49 different participant bikers, and 128,083 GPS points. A second real dataset is obtained from [28] (MS-dataset), which contains GPS-tracks from 182 users in a period of over five years collected by researchers at Microsoft. Although this dataset contains 17,621 trajectories, we pruned long trajectories (>2 hours) and selected only the ones within a selected rectangular bound (near Beijing metro area). This reduced the dataset to 169 users and 5,887 trajectories. To demonstrate the scalability of our approach, we also used a larger synthetic dataset (MNTG-dataset) generated using Minnesota Web-based Traffic Generator [14]. The generated MNTG-dataset contains 3,500 moving objects, having 10 trajectories each, while we set the available options such that the objects were placed randomly—not constrained by the underlying network. For every data object in the synthetic dataset, we generated its weight uniformly in the range from 1 to 50.

Machine and Tools: We implemented our proposed 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. **Measurements and Default Values:** For each of the dataset used in the experiments, we considered one trajectory per object at a run. We performed 20 such runs for each experiment (unless explicitly indicated otherwise) to get more valid representative-observations. The default values of the number of objects for BIKE, MS, and MNTG dataset are 49, 169, and 3500 respectively. The query time is set to the whole time-period (lifetime of trajectories) during a particular run for all the datasets, and the base value of range area for BIKE, MS, and MNTG dataset is 500000, 100000, and 400000 m^2 respectively (denoted X). We note that all the datasets and the source code of the implementations are publicly available at: <http://www.cs.northwestern.edu/~mmh683/projects-works/>.

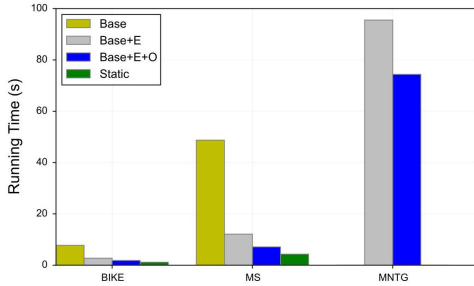


Figure 8: Running-time in different datasets.

5.2 Results

We now discuss the experimental results, starting with the running time of different algorithms, and the errors obtained by Static-MaxRS (with respect to Co-MaxRS). Finally, we evaluate the performance of pruning strategies, and analyze the impact of cardinality and the range (R) size.

5.2.1 Running Time Comparison

We ran the algorithms over the three datasets and the result is shown in Figure 8: *Base*, (*Base+E*), (*Base+E+O*),

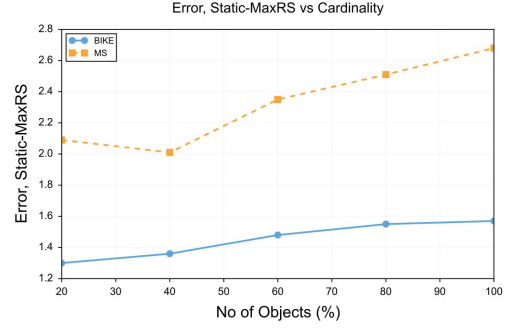


Figure 9: Cardinality impact on Static-MaxRS error.

and *Static* denote the base Co-MaxRS, base+events pruning, base+both events and objects pruning, and Static-MaxRS algorithms, respectively. 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 (preventing unnecessary recomputations). In the case of MNTG-dataset, we used a query time-period of 60 seconds instead of the whole trajectory-lifetime. For this reason, we omitted the data of Static-MaxRS for this dataset as its running time is not dependent on query time (rather than it depends on the number of objects and intersections among them). On the other hand, even in this restricted setting for MNTG-dataset, running time of base Co-MaxRS was very large (approximately 980s) – thus, not included in Figure 8 as it will skew the graph. Naturally, as the number of objects increases, so does the running times of all the algorithms.

5.2.2 Error in Static-MaxRS

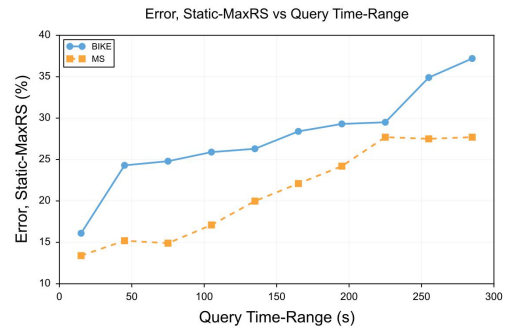


Figure 10: Duration impact on Static-MaxRS error.

We now illustrate the errors induced by using Static-MaxRS to approximate Co-MaxRS. In Figure 9, the behavior of the error with increasing the number of objects is shown over both the real-world datasets. As cardinality of the dataset increases, the absolute error also grows, albeit slowly. Note that, we exclude performing Static-MaxRS related experiments on the large synthetic dataset (MNTG-dataset), as the correctness, rather than scalability, is a concern. We detect similar trend when comparing the induced error against growing query time-range in Figure 10. As the query time-period expands, the error-percentage grows at a

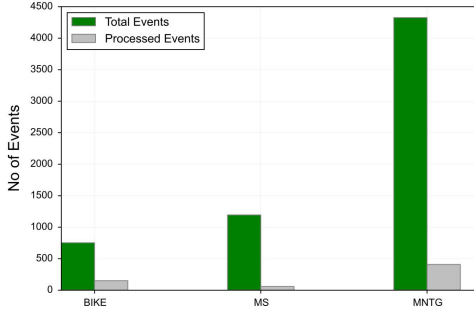


Figure 11: Events pruning over different datasets.

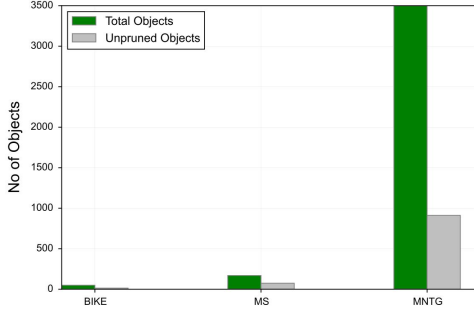


Figure 12: Objects pruning over different datasets.

rapid rate. This shows that although Static-MaxRS has its own merits, it is not suitable to use in place of Co-MaxRS for mobile objects. In the next parts of the experimental observations, we mainly focus on the benefits of pruning strategies in all the datasets.

5.2.3 Performance of Pruning Strategies

Figure 11 illustrates 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, we can also deduce that the number of events grows as the dataset becomes larger, but the number of actual recomputation-events are well within the worst-case theoretical upper-bound. Similar results are obtained for the objects pruning scheme, as demonstrated in Figure 12 – indicating that the pruning schemes perform nearly equally well in all three datasets.

5.2.4 Impact of Cardinality

Figure 13 illustrates the impact of the cardinality on our pruning methods. In Figure 13a, from the experiment done on the BIKE-dataset, we can deduce an interesting relation: as the dataset increases, more \vec{OD} kind of events are pruned, whereas (cf. Figure 13b), objects pruning slightly decreases for \vec{OD} as the dataset increases. On the other hand, \vec{DO} events exhibit completely opposite behavior. This, in a sense, neutralizes the overall impact of the increase in cardinality for our pruning scheme. Figure 13c 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.

5.2.5 Influence of Range Size

The final 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 we found out in Figure 14, increasing range area results in fewer portion of events pruned. This occurs because as the area of R grows, there is more probability of overlapping rectangles within the moving objects. Similarly, the growing rectangle size had adverse effects on the objects pruning scheme as well. But, in practice, the area of R should be quite smaller than the overall bounding space. Even with quite large values of R (e.g., 50000 m^2) we have more than 50-60% of pruning through our methods.

6. RELATED WORKS

There are several bodies of research results that are closely related, and were used as foundation throughout our work.

As mentioned, the problem of MaxRS was first studied in the Computational Geometry community, and [11] proposed an in-memory algorithm to find a maximum clique of intersection graphs of rectangles in the plane. Subsequently, [16] devised a new algorithm based on the 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 of these works provide theoretically optimal bound, they are not practically suitable to be directly applied in large spatial databases. In this work, we used the method of [16] to recompute MaxRS only at certain KDS events, however, we proposed pruning strategies to reduce the number of such invocations.

The MaxRS problem in spatial databases was investigated in [5], where a scalable external-memory algorithm that is optimal in terms of the I/O complexity was proposed. Extended versions of this work also deal with $(1 - \epsilon)$ -approximate MaxRS and All-MaxRS problems [6]. Essentially, [5] and [6] divide the space recursively into m vertical slabs until the count of objects within a slab is such that it can be processed in memory. However, in the context of spatio-temporal data, due to the movement of objects between different slabs, a slab which has few objects at one time, can have a lot of objects at a later time—which may render the processing of the objects within a given slab impossible in memory. In other words, any kind of static sub-division of the space is likely not to be useful in spatio-temporal databases.

Recent works have investigated variants of the MaxRS problem, e.g., in [19] 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. Although both [19] and [4] deal with couple of interesting variants of the traditional MaxRS problem, they do not consider the settings of mobile objects. More recently, [10] demonstrated an implementation of a system that uses an in-network algorithm to process MaxRS queries in wireless sensor networks (WSN). Specifically, the individual static sensor nodes were considered as objects for which the weights corresponded to the values of the underlying sensed phenomenon (e.g., light, temperature, etc.). In these settings, the weights of the objects may change with time, although the location is static throughout.

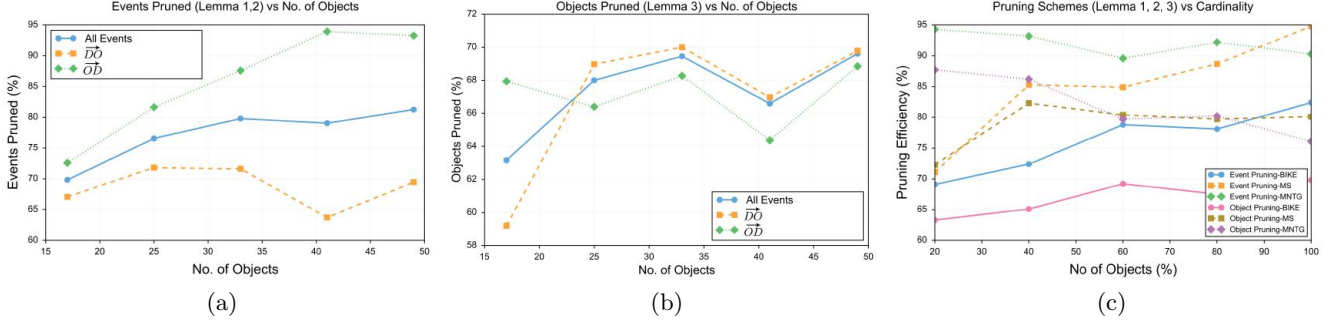


Figure 13: Impact of cardinality on the pruning schemes: (a) Different events pruning (BIKE-dataset) (b) Objects pruning (BIKE-dataset) (c) Overall objects and events pruning (all datasets).

In this work, we did rely on the concept of a kinetic data structure (KDS) framework introduced and practically evaluated in [1] and [2]. The KDS-like data structure was used to process critical events at which the current MaxRS solution may change. To measure the quality of a KDS, both [1] and [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—and we used the same measures to evaluate the quality of our approach.

7. CONCLUSION AND FUTURE WORKS

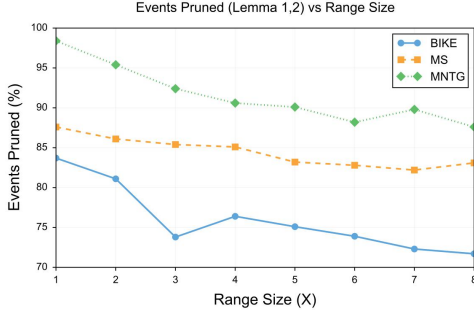


Figure 14: Effectiveness of events pruning strategy against varying range sizes.

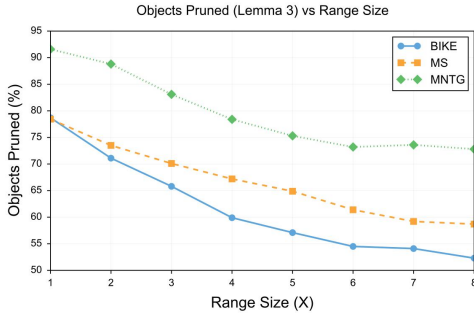


Figure 15: Effectiveness of objects pruning scheme against varying range sizes.

We addressed a novel spatio-temporal variant of the MaxRS problem – determining the location of a given rectangle R that covers the maximum number of points from a given set – in the setting of moving objects trajectories. In contrast to the MaxRS problem first studied by the computational geometry community [11, 16], the Continuous MaxRS (Co-MaxRS) solution may change with time. As a “transition” step, we analyzed the Static-MaxRS variant where a static position for the range (a rectangle R) is picked for the entire duration of interest, with maximum number of moving objects intersecting R at some time instant. However, this variant does not guarantee maximality at every time instant. 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 – i.e., when the collection of objects constituting the (current) answer changes. These critical points (event points) occur when the “dual-rectangles” (i.e., rectangles r_i identical to R centered at each moving object o_i) changed their topological relationship. To maintain the needed information at each event point, we used a Kinetic Data Structures (KDS) paradigm, where information was updated at each event point. While the sequence of all the transitions (overlap-to-disjoint ($\vec{O}\vec{D}$) and disjoint-to-overlap ($\vec{D}\vec{O}$)) defines the upper-bound, we also proposed two pruning heuristics – one to eliminate events from KDS and one to eliminate the number of actual objects – when the re-computation of the Co-MaxRS can be avoided. We also demonstrated that, while our algorithms focused on the moving objects (resp. rectangles), 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, demonstrated that the heuristics enabled significant speed-ups in terms of the overall computation time from the upper bound on the time complexity.

There are numerous avenues of extending our work. In the spirit of [5, 6], one task is to devise a suitable indexing structure that will minimize the I/O overheads when trajectories data sets is large enough and needs to reside on a secondary storage or even on cloud [7]. Similarly, we plan to investigate the trade-offs between the gains in processing time vs. approximate answer to Co-MaxRS – a challenge of its own being to properly formalize the notion of “approximate” in continuous settings. A specific example of the last claim stems from the fact that in our solution, there may

be cases where Co-MaxRS has discontinuities – i.e., the current MaxRS needs to instantaneously change its location. Clearly, one may want to sacrifice the precision of the answer for the benefit of having a realistic time-budget for the MaxRS to “travel” from one such location to another. Complementary to this is the investigation of the k -variant of Co-MaxRS – i.e., the case of having multiple mobile cameras whose fields of view should jointly guarantee a maximal coverage of mobile entities.

8. REFERENCES

- [1] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.
- [2] J. Basch, L. J. Guibas, C. D. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, pages 388–390. ACM, 1997.
- [3] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.
- [4] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, X. Cheng, and P. Chen. Rotating MaxRS queries. *Information Sciences*, 305:110–129, 2015.
- [5] D.-W. Choi, C.-W. Chung, and Y. Tao. A scalable algorithm for Maximizing Range Sum in spatial databases. *Proceedings of the VLDB Endowment*, 5(11):1088–1099, 2012.
- [6] 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.
- [7] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce framework for spatial data. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1352–1363, 2015.
- [8] R. H. Güting and M. Schneider. *Moving objects databases*. Elsevier, 2005.
- [9] F. J. Harvey and K. J. Krizek. Commuter bicyclist behavior and facility disruption. Technical report, 2007.
- [10] M. M. Hussain, P. Wongse-ammatt, and G. Trajcevski. Demo: Distributed MaxRS in wireless sensor networks. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 479–480. ACM, 2015.
- [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):310–323, 1983.
- [12] 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):12990–13005, 2014.
- [13] 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.
- [14] 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*, pages 38–55. Springer, 2013.
- [15] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-temporal access methods. *IEEE Data Eng. Bull.*, 26(2):40–49, 2003.
- [16] 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):45–61, 1995.
- [17] J. Ni and C. V. Ravishankar. PA-tree: A parametric indexing scheme for spatio-temporal trajectories. In *Advances in Spatial and Temporal Databases*, pages 254–272. Springer, 2005.
- [18] T. J. Ottosen and J. Vomlel. Honour thy neighbour—clique maintenance in dynamic graphs. In *Proceedings of the Fifth European Workshop on Probabilistic Graphical Models (PGM-2010)*, pages 201–208, 2010.
- [19] 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.
- [20] J. B. Rocha-Junior, A. Vlachou, C. Doukeridis, and K. Nørvang. Efficient processing of top- k spatial preference queries. *Proceedings of the VLDB Endowment*, 4(2):93–104, 2010.
- [21] J. Schiller and A. Voisard. *Location-based services*. Elsevier, 2004.
- [22] S. Shekhar and S. Chawla. *Spatial databases: A tour*, volume 2003. prentice hall Upper Saddle River, NJ, 2003.
- [23] V. Stix. Finding all maximal cliques in dynamic graphs. *Computational Optimization and applications*, 27(2):173–186, 2004.
- [24] L. Wang, Y. Zheng, X. Xie, and W. Y. Ma. A flexible spatio-temporal indexing scheme for large-scale GPS track retrieval. In *Mobile Data Management, 2008. MDM’08. 9th International Conference on*, pages 1–8. IEEE, 2008.
- [25] F. Wu, T. K. H. Lei, Z. Li, and J. Han. MoveMine 2.0: Mining object relationships from movement data. *Proceedings of the VLDB Endowment*, 7(13):1613–1616, 2014.
- [26] X. Xiao, B. Yao, and F. Li. Optimal location queries in road network databases. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 804–815. IEEE, 2011.
- [27] Y. Zheng. Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29, 2015.
- [28] Y. Zheng, L. Zhang, X. Xie, and W. Y. Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800. ACM, 2009.
- [29] Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu. MaxFirst for MaxBR k NN. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 828–839. IEEE, 2011.