Maximizing Area-Range Sum for Spatial Shapes (MAxRS^3)*

Muhammed Mas-ud Hussain
Northwestern University
Evanston, IL, USA
mas-ud@u.northwestern.edu

Goce Trajcevski
Iowa State University
Ames, IA, USA
gocet25@iastate.edu

ABSTRACT

We investigate a novel variant of the well-known MaxRS (Maximizing Range Sum) problem – namely, the MAxRS^3 (Maximizing Area-Range Sum for Spatial Shapes). The MaxRS problem amounts to detecting a location where a fixed-size rectangle R should be placed, so that it covers a maximum number of points – or sum of weights, if the points are weighted – from a given input set of 2D points. While variants have tackled the settings in which the input set to MaxRS problem consists of polygons instead of points – the solution is still based on (weighted) count. We postulate that in many practical applications it is of interest to determine where to place the input rectangle so that the total area-coverage in its interior is maximized. In this paper, we formalize the MAxRS^3 problem and propose (to our knowledge) the first solution to this new problem.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**; *Database query processing*;

KEYWORDS

Maximizing Range Sum, Spatial Shapes, MAxRS^3

ACM Reference Format:
Muhammed Mas-ud Hussain and Goce Trajcevski. 2018. Maximizing Area-Range Sum for Spatial Shapes (MAxRS^3). In *SSDBM '18: 30th International Conference on Scientific and Statistical Database Management, July 9–11, 2018, Bozen-Bolzano, Italy.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3221269.3221301

1 INTRODUCTION

The Maximizing Range Sum query (MaxRS) takes a collection of weighted spatial point-objects O and a rectangle R with fixed dimensions as inputs, and generates a location(s) for placing the centroid of R that maximizes the sum of the (weights of the objects) in R's interior. Initially, the MaxRS problem was identified and solved by the researchers in computational geometry (CG) community [6]. Some years later, motivated by its importance in location-aware

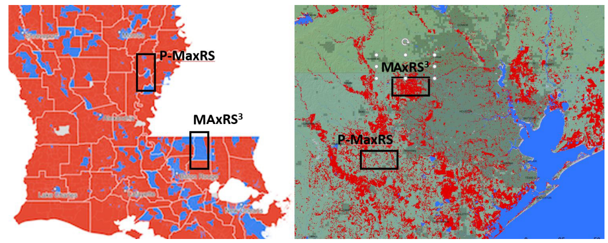

Figure 1: MAxRS^3 – Louisiana votes and Texas floods.

queries, such as: *what is the best location for a new franchise store with a limited delivery range*, or *what is the hotel location so that a tourist with spatially constrained mobility can see most attractions* – researchers have tackled various new aspects. Efficient solution for MaxRS in large (secondary storage) spatial databases has been presented in [3]; more recently, a continuous variant of MaxRS for mobile objects and query-rectangle has been addressed in [4], and dynamic settings where objects may be inserted/deleted along with changing their weights have also been considered (cf. [1, 5]).

We note that [6] considered a variation of the MaxRS problem where the input collection consists of polygons instead of points. For brevity, we call that variant a P-MaxRS (Polygons MaxRS), and a solution was presented so that the (weighted) sum of the polygons inside R is maximized.

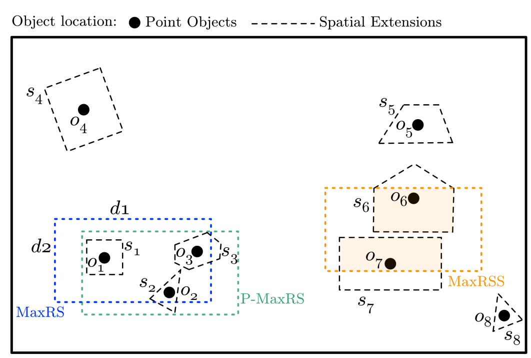

Figure 2: MaxRS vs. P-MaxRS vs. MAxRS^3.

What motivates our work is the observation that in many practical settings, in addition to the datasets consisting of polygons – it is more important to find a placement for the centroid of query-rectangle R in a manner that will ensure *maximal area coverage*. As specific examples, consider the following scenarios:
S1: A campaign manager with a limited reachability for his staff would like to know where to place the mobile headquarters to improve the votes in a given region.

*Research supported by NSF grants III 1213038 and CNS 1646107, ONR grant N00014-14-10215 and HERE grant 30046005.

SSDBM '18, July 9–11, 2018, Bozen-Bolzano, Italy

ACM ISBN 978-1-4503-6505-5/18/07...$15.00
https://doi.org/10.1145/3221269.3221301

S2: Emergency crews are interested in location for placing the sump-pumps with limited reachability of multiple hose, so that the drainage impact is maximized.

Both scenarios are illustrated in Fig. 1 (left portion is illustrating **S1** and the right portion illustrates **S2**). In each case, we show two positions of R: (1) covering maximal number of regions (i.e., P-MaxRS); and (2) covering maximal area. Clearly, the placement(s) based on the solution to P-MaxRS are not the desired output for **S1** and **S2**. To address such problems, in this paper we propose the MAxRS^3 (Maximal Area-Range Sum for Spatial Shapes) problem. More formally, the fundamental differences between MaxRS, Polygon Containment (a.k.a. P-MaxRS) and MAxRS^3 problem are illustrated in Figure 2. Assuming that 8 point objects are given ($o_1, o_2, \ldots, o_8$ in Figure 2) and the weights of all the objects are uniform, the placement of the rectangle R indicated in dotted blue line is the MaxRS solution (i.e., count=3). When objects have a spatial extent represented by a polygon, e.g., triangle, rectangle, pentagon, etc. (cf. $s_1, s_2, \ldots, s_8$ centered at the point objects $o_1, o_2, \ldots, o_8$ in Figure 2), the solution to the P-MaxRS problem [6] is given by the dotted green line, where the placement of R completely encapsulates polygons s_1, s_2, and s_3 (i.e., count=3). The MAxRS^3 problem – addressing the more practical goal of maximizing the area of the coverage of R, will return the placement represented by the dotted orange line, overlapping s_6 and s_7.

2 PROBLEM FORMULATION

We now review the basics of sweep-line technique and the standard approach for solving MaxRS and P-MaxRS problem, and subsequently, formally introduce the MAxRS^3 problem.

Fundamentally, both are based on a sweep-line technique [2, 7] – a paradigm of conceptually "sweeping" a horizontal (or vertical) line across the plane, stopping at certain discrete points (called *events*) to perform different tests/computations. The events are marked by corresponding Y-coordinates (for horizontal sweep-line) or X-coordinates (for a vertical sweep-line) at which "something interesting" happens.

At each event $e_i \in E$, some geometric computations need to performed with the objects that either intersect or are in the immediate vicinity of the sweep line, and the final solution is available once the line has passed over all objects.

In general, sweep-line algorithms maintain a data structure to store the events, generally sorted by X or Y coordinates, and at a given instance, the data structure stores only the active events. The overall processing time for a sweep-line algorithm is $O(|E| \times P_{e_i})$, where $|E|$ = total number of events, and P_{e_i} = the processing time of each event. Thus, when designing a sweep-line technique, the goal is to minimize $|E|$ and P_{e_i}.

MaxRS for Point Objects: Let $C(p, R)$ denote the region covered by an isothetic rectangle R, placed at a particular point p. Given a rectangular spatial field $\mathbb{F}$, an axis-parallel rectangle R (of size $d_1 \times d_2$), and a set O of n spatial points $O = \{o_1, o_2, \ldots, o_n\}$ (bounded by $\mathbb{F}$), where each o_i is associated with 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. If $\forall o_i \in O : w_i = 1$, we have the *count* variant (cf. Figure 2). An in-memory solution to MaxRS (cf. [6]) 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 .

Using this transformation, [6] proposed a sweep-line algorithm to solve the MaxRS problem. 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 horizontal line is swept vertically. The line stops at the top and bottom edges of each rectangle (a.k.a. events). During each event ($|E|$=$2n$), the interval tree is updated accordingly, and the count (i.e., the number of overlapping rectangles) for each interval currently residing in the tree is computed (P_e=$O(\log n)$). An interval with the maximum count during the entire process is returned as the final solution and, the algorithm takes $O(n \log n)$ (i.e., $O(|E| \times P_{e_i})$) time.

MaxRS for Polygons: [6] proposed an extension that considers polygons instead of point objects. The problem addressed in [6] (**(P-MaxRS)**) is: Given a rectangular spatial field $\mathbb{F}$, an axis-parallel rectangle R (of size $d_1 \times d_2$), and a set S of n non-overlapping spatial regions (convex polygons) $S = \{s_1, s_2, \ldots, s_n\}$ (bounded by $\mathbb{F}$), the answer to P-MaxRS query ($\mathbb{A}_{P\text{-}MaxRS}(S, R)$) retrieves a position p for placing the center of R, such that:

$$\sum_{\{\forall s_i \in S\}} \begin{cases} 1, & \text{if } (s_i \cap C(p,R)) = s_i \\ 0, & \text{otherwise} \end{cases}$$

is maximal.

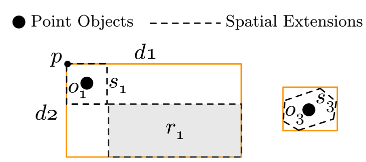

Figure 3: P-MaxRS processing scheme.

$(s_i \cap C(p, R)) = s_i$ ensures that s_i is fully enclosed within R. At first, a base solution is devised assuming all $s_i \in S$ are axis-parallel rectangles. If any given s_i is larger than R, it can safely be pruned, i.e., it cannot be fully enclosed by R. For a polygon s_i, we have to place R with its top-left corner at p, where p is the top-left corner of s_i (cf. s_1 in Figure 3). Suppose, r_i is the rectangle drawn from the bottom-right point of s_i – e.g., r_1 in Figure 3. Clearly, R will enclose s_i completely, if and only if the bottom-right corner of R lies in r_i, which is defined as the *prime rectangle* for s_i. Given the prime rectangles of all axis-parallel rectangles $s_i \in S$, the problem can be converted to the rectangle intersection problem. In case of arbitrary polygons, R encloses a polygon if and only if R encloses its minimum bounding rectangle (MBR) as shown in Figure 3 for s_3. Thus, given MBR s'_i for all $s_i \in S$, the same techniques for axis-parallel rectangles can be applied here too.

MAxRS^3: In many practical scenarios, maximizing the overall coverage area over the given set of polygons is of more importance than the P-MaxRS problem which can return a placement with many small polygons (see Figure 2). For example, suppose the set of given polygons represent flood-affected spatial regions within a state/country. The objective then is to find a way to maximize aid support by reaching to as large amount of flood-affected area as possible. Let us use $A(r)$ to denote the area of a given region r. Based on these observations, we introduce a novel problem Maximizing Area-Range Sum for Spatial Shapes (MAxRS^3) as follows: Given a rectangular spatial field $\mathbb{F}$, an axis-parallel rectangle R (of size $d_1 \times d_2$), and a set S of n non-overlapping spatial regions (convex polygons) $S = \{s_1, s_2, \ldots, s_n\}$ (bounded by $\mathbb{F}$), the answer to MAxRS^3 query ($\mathbb{A}_{MAxRS^3}(S, R)$) retrieves a position p for placing the center of R, such that $\sum_{\{\forall s_i \in S\}} A\,(s_i \,\cap\, C(p, R))$ is maximal.

We term $\sum_{\{\forall s_i \in S\}} A\,(s_i \,\cap\, C(p, R))$ as the *score* of any position p for MAxRS^3 problem. We note that both P-MaxRS and MAxRS^3 only consider non-overlapping polygons. If there is overlap between two polygons s_i and s_j, we can either: (1) Combine the two polygons into a new single one, e.g., $s_{new} = s_i \cup s_j$; or, (2) Consider three separate disjoint polygons $s_k (= s_i \cap s_j)$, $s_i - s_k$, and $s_j - s_k$. The ideas presented in this paper can be readily extended to include concave (non self-intersecting) polygons, but for brevity we keep the discussion and algorithms limited to convex case.

3 PROCESSING MAXRS^3

We now discuss the challenges relevant to processing MAxRS^3, and devise an efficient algorithm by using a pair of top-to-bottom sweep-lines, accompanied by two left-to-right sweep-lines.
Although MaxRS and P-MaxRS have efficient $O(n \log n)$ solutions, processing MAxRS^3 poses a different set of challenges:
• Both MaxRS and P-MaxRS can be transformed into rectangle intersection problem. Same is not true for MAxRS^3 since containing the polygons "completely" is not required. The solution for MAxRS^3 considers the area to be included to compute the maximal coverage.
• The discrete event-points need to be identified, along with the corresponding processing at each "interesting" points.

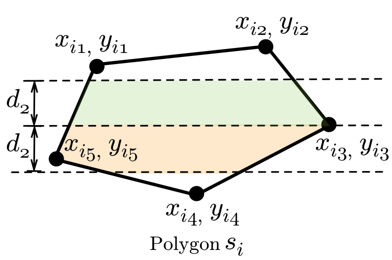

Figure 4: Covered area and vertices of a given s_i.

Discrete Points: To identify the events, let us assume that each polygon $s_i \in S$ consists of m_i vertices – $v_{i1}, v_{i2}, \ldots, v_{im_i}$, where $v_{i1} = (x_{i1}, y_{i1})$, $v_{i2} = (x_{i2}, y_{i2})$, ... $v_{im} = (x_{im}, y_{im})$, forming m_i edges by connecting adjacent vertices in a pair-wise manner, e.g., $[\{v_{i1}, v_{i2}\}, \{v_{i2}, v_{i3}\}, \ldots, \{v_{im_i}, v_{i1}\}]$. From the setting of MAxRS^3, we observe that the area of a polygon covered by R can always be decomposed into a trapezoid (rectangle and squares are a special case of trapezoid) or a triangle. In both cases, the covered area is a function of base (i.e., length of edges) and height – which depends on the slope of certain edges. However, we observe that the slope of any given polygon s_i changes only at the vertices, i.e., $v_{i1}, v_{i2}, \ldots, v_{im_i}$ (see Figure 4). Thus, we use vertices of the input polygons as our discrete event-points.

Multiple Sweep-lines: At first, we propose to sweep the space in top-to-bottom manner, i.e., via using a horizontal sweep-line. During each event at a vertex v_{ij}, we have to compute maximal placement of R having the highest coverage in the vicinity of v_{ij}. An interesting observation is that the optimal placement of R may cover both v_{ij}'s above (i.e., up to Y-axis coordinate $(y_{i3} + d_2)$, green in Figure 4) or below (i.e., up to Y-axis coordinate $(y_{i3} - d_2)$, orange in Figure 4) regions. Thus, we use two sweep-lines in our algorithm, always maintaining a Y-axis distance of d_2 between them. Let us consider an example scenario presented in Figure 5, where there are 4 polygons considered: s_1, s_2, s_3 and s_4. The discrete points of interest will be all the vertices v_{ij}, e.g., $v_{11}, v_{21}, v_{31}, v_{41}$, etc. In total, there are 18 such vertices in this setting. At first we will sweep the space in top-to-bottom manner, and use two horizontal sweep-lines: (1) a leader horizontal line (l_h); and (2) a follower horizontal line (f_h). During the whole process, l_h leads (i.e., is below) f_h in the sweeping and the distance between f_h and l_h is set to d_2, i.e., $Y_{l_h} = Y_{f_h} - d_2$ (assuming (0,0) is bottom-left point), where Y_{l_h} and Y_{f_h} denote the Y-coordinate values of the corresponding sweep-lines. Both l_h and f_h stop at all vertices v_{ij} during the sweep – thus, we have two kinds of events: (1) e_{hl}, when the leader horizontal line l_h stops at a vertex; and (2) e_{hf}, when the follower horizontal line f_h stops at a vertex (see Figure 5). In total, there will be $18 \times 2 = 36$ events in the example provided in Figure 5.

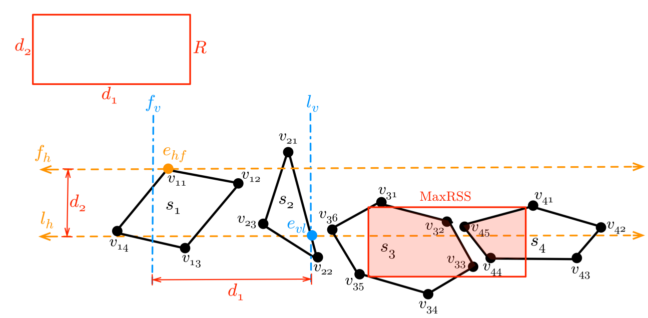

Figure 5: Leader and follower sweep-lines for MAxRS^3.

Events Processing Scheme: In case of e_{hl} and e_{hf} events, we will only consider the space bounded by the two horizontal lines l_h and f_h, i.e., $[Y_{l_h}, Y_{f_h}]$. For example, in Figure 5, for an event e_{hf} at v_{11}, only the region bounded by the two orange lines will be explored. We will again use the concept of multiple sweep-lines to compute the placement having highest *score* in $[Y_{l_h}, Y_{f_h}]$ bounded region, but this time, sweeping will take place in a left-to-right manner. The idea is to use two vertical sweep-lines: (1) a leader vertical line (l_v); and (2) a follower vertical line (f_v) (blue lines in Figure 5). Similar to the horizontal sweep-lines, l_v leads (i.e., is on the right of) f_v and the distance between f_v and l_v is set to d_1, i.e., $X_{l_v} = X_{f_v} + d_1$, where X_{l_v} and X_{f_v} denote the X-coordinate values. For this secondary sweeping process, the discrete points of interest are the current intersection points between the polygons and the horizontal lines – l_h and f_h. Both l_v and f_v stop at all such intersection points during the left-right sweep – thus, we have two kinds of events: (1) e_{vl}, when l_v is involved; and (2) e_{vf}, when f_v is involved. For example, the vertical leader event e_{vl} (blue circle) occurs at one of the intersection points of s_2 and l_h during the horizontal follower event e_{hf}. There will be $11 \times 2 = 22$ such events in the example provided in Figure 5.

During each e_{vl} and e_{vf}, we have to find the location p^* having highest *score*, i.e., $\sum_{\{\forall s_i \in S\}} A\,(s_i \cap C(p^*, R))$. For this, we use the fact that the covered area of a given polygon via R can always be decomposed into a trapezoid or triangle. We can compute the area of a given polygon (such as s_i in Figure 4, where m_i=5) as follows:

$$A(s_i) = \frac{1}{2} \times \left(\begin{vmatrix} x_{i1} & y_{i1} \\ x_{i2} & y_{i2} \end{vmatrix} + \begin{vmatrix} x_{i2} & y_{i2} \\ x_{i3} & y_{i3} \end{vmatrix} + \ldots + \begin{vmatrix} x_{im_i} & y_{im_i} \\ x_{i1} & y_{i1} \end{vmatrix} \right) \quad (1)$$

When performing the left-to-right sweeping:

(1) The intersection points and covered portion of edges of the polygons change. For every little increment of covered portion δ, we already have (or, can pre-compute) the constant slopes of the respective edges. In Equation 1, all x_{ij} and y_{ij} values are constants.

(2) We need to maximize A in Equation 1 with the optimal placement of R during an event, within the "permissible" ranges of δ. However, given the range for δ, Equation 1 is a sum of quadratic functions in δ and its first derivative is a linear one – thus, the extreme can be calculated analytically.

(3) Most importantly, the ranges for δ are always bounded by the event-points of both horizontal and vertical sweep-lines, i.e., e_{hf}, e_{hl}, e_{vf}, and e_{vl}. Thus, we can compute p^* for R during a vertical line-event, even if p^* is in somewhere between two consecutive events. In summary, we perform a top-to-bottom sweep-line technique using two horizontal lines l_h and f_h, and then, perform a left-to-right sweep of the bounded space by two vertical lines l_v and f_v at each e_{hf} or e_{hl}. During the whole process, we keep track of the maximal coverage area and placement p^*, and eventually, return the result at the end of the top-to-bottom sweeping.

Algorithmic Details: The processing of MAxRS^3 is formalized in Algorithm 1. In line 1, vertices of all the polygons are sorted in order of their y_{ij} value and inserted into a list v_{list}. In lines 2 – 5, relevant variables are initialized. Lines 6 – 18 constitute the main working loop, i.e., the top-to-bottom sweeping. Lines 7 – 14 check whether the next event should be e_{hl} or e_{hf}, and variables are updated accordingly. Line 15 performs the left-to-right sweeping using l_v and f_v. For brevity, we skip the details of this secondary sweeping in Algorithm 1. The maximal coverage area and optimal placement is tracked via lines 16 – 18.

Complexity: Let us assume that there are n polygons, with m vertices each – so sorting in line 1 of Algorithm 1 takes $O((n \times m)\log(n \times m))$. There will be $O(n \times m)$ horizontal line events, i.e., e_{hl} or e_{hf} – (cf. lines 6 – 18 of Algorithm 1). In each such event, when the left-to-right sweep starts, there can be at most $2 \times 2 = 4$ intersections per non-overlapping convex polygons with l_h and f_h, i.e., $O(n)$ intersections in total. The area-calculation needs $O(m)$ number of 2×2 determinants per polygon, taking a worst-case total cost of $O(n \times m)$ at each e_{hl} or e_{hf} event. Thus, the overall time complexity is $O((n \times m)^2)$.

Algorithm 1: ProcessMAxRS^3 $(S, R, \mathbb{F})$

Input : A set of non-overlapping convex polygons S, query rectangle R of size $d_1 \times d_2$ and bounding box $\mathbb{F}$

Output : $\mathbb{A}_{MAxRS^3}$ (i.e., p^*)

1 $v_{list} \leftarrow$ the list of all vertices v_{ij} of each polygon $s_i \in S$ sorted by their y_{ij} value;
2 $Y_{l_h} \leftarrow \mathbb{F}.height$;
3 $Y_{f_h} \leftarrow \mathbb{F}.height + d_2$;
4 $e_{hl}\_index, e_{hf}\_index \leftarrow 0$;
5 $p^*, max\_coverage\_area \leftarrow NULL, 0$;
6 **while** $e_{hl}\_index < |v_{list}|$ ***or*** $e_{hf}\_index < |v_{list}|$ **do**
7   **if** $(Y_{l_h} - v_{list}[e_{hl}\_index].y_{ij}) \leq (Y_{f_h} - v_{list}[e_{hf}\_index].y_{ij})$ **then**
8     $Y_{l_h} \leftarrow v_{list}[e_{hl}\_index].y_{ij}$;
9     $Y_{f_h} \leftarrow Y_{l_h} + d_2$;
10     $e_{hl}\_index \leftarrow e_{hl}\_index + 1$;
11   **else**
12     $Y_{f_h} \leftarrow v_{list}[e_{hf}\_index].y_{ij}$;
13     $Y_{l_h} \leftarrow Y_{f_h} - d_2$;
14     $e_{hf}\_index \leftarrow e_{hf}\_index + 1$;
15   $p^{local}, local\_coverage\_area \leftarrow$ Perform left-to-right sweep using vertical lines l_v and f_v;
16   **if** $local\_coverage\_area > max\_coverage\_area$ **then**
17     $max\_coverage\_area \leftarrow local\_coverage\_area$;
18     $p^* \leftarrow p^{local}$;
19 **return** p^*

4 CONCLUDING REMARKS

We introduced a novel variant of the MaxRS problem – the MAxRS^3 problem, which determines the placement for a given fixed rectangle R in a 2D plane, such that the sum of the areas of the intersections of a (subset of a) given collection of polygonal shapes and R is maximized. We also presented the solution to MAxRS^3 along with the corresponding algorithmic implementation. Presently, we are investigating techniques for pruning certain events from consideration during the sweep-line process to speed up the execution. We are also working on the scalability aspect – i.e., access structures for the cases when the input is too large to fit in the main memory.

REFERENCES

[1] Daichi Amagata and Takahiro Hara. 2016. Monitoring MaxRS in Spatial Data Streams. In *19th EDBT*.
[2] Ulrike Bartuschka, Kurt Mehlhorn, and Stefan Näher. 1997. A robust and efficient implementation of a sweep line algorithm for the straight line segment intersection problem. In *IN PROC. WORKSHOP ON ALGORITHM ENGINEERING*. Citeseer.
[3] D. W. Choi, C. W. Chung, and Y. Tao. 2014. Maximizing Range Sum in External Memory. *ACM Trans. Database Syst.* 39, 3 (Oct. 2014), 21:1–21:44.
[4] Muhammed Mas-ud Hussain, Goce Trajcevski, Kazi Ashik Islam, and Mohammed Eunus Ali. 2017. Visualization of Range-Constrained Optimal Density Clustering of Trajectories. In *SSTD*. 427–432.
[5] Mir Imtiaz Mostafiz, S. M. Farabi Mahmud, Muhammed Mas-ud Hussain, Mohammed Eunus Ali, and Goce Trajcevski. 2017. Class-based Conditional MaxRS Query in Spatial Data Streams. In *SSDBM*.
[6] Subhas C Nandy and Bhargab B Bhattacharya. 1995. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications* 29, 8 (1995).
[7] Michael Ian Shamos and Dan Hoey. 1976. Geometric Intersection Problems. In *FOCS*.