Continuous Maintenance of Range Sum Heat Maps

Jianzhong Qi #1**, Vivek Kumar #2, Rui Zhang #1*, Egemen Tanin #1, Goce Trajcevski †, Peter Scheuermann ††

# *School of Computing and Information Systems, The University of Melbourne, Australia*

1 {jianzhogn.qi, rui.zhang, etanin}@unimelb.edu.au, 2 vivekk1@student.unimelb.edu.au

† *Department of Electrical and Computer Engineering, Iowa State University, USA*

gocet25@iastate.edu

†† *Department of Electrical Engineering and Computer Science, Northwestern University, USA*

peters@eecs.northwestern.edu

***Abstract*—We study the problem of *continuous maintenance of range sum heat maps* over dynamically updating data objects. The *range sum* (RS) here refers to the sum of the weights of the data objects enclosed by a given range (rectangle) R. Range sum problems are useful in spatio-temporal data analytics and decision making processes. Recent studies on range sum problems focus on computing the *MaxRS* query, which finds a location to place a rectangle R such that its RS is maximized. In real applications, knowing only the location with the maximum RS may be insufficient, because decision making is a multi-factor process where maximizing the RS may just be one of the factors. It is also important to gain an overview of the RS distribution at different locations, so that decisions can be made based on global knowledge. We therefore propose to compute a range-sum heat map that visualizes the RS value for every location in a data space. Considering that data objects may be inserted into or removed from the data space dynamically, we further study the continuous maintenance of range-sum heat maps over dynamically updating data objects. We adapt algorithms to compute range-sum heat maps and to perform heat map updates. We build a demo system to showcase the usefulness of range sum heat maps and the effectiveness of the adapted algorithms.**

I. Introduction

Range sum problems [1], [2] aim to compute the range sum (RS) that is the sum of the weights of the objects enclosed by a given range which is usually modeled by a rectangle R. Such problems have useful applications in spatio-temporal data analytics. A typical application is to help the decision making process for resource allocations. For example, a range sum problem can compute the number of mobile users in the city CBD as the RS, where the mobile users are the data objects (each with a weight of 1) and the city CBD is the range R. This RS can help mobile service providers decide whether additional base stations are needed to serve the users. As another example, consider a scenario where the administration of a national park needs to deploy some supplies (e.g., water or medicine) for the wild animals in the park due to a drought or an epidemic. Suppose that the animals are tagged with sensors and hence their locations are known. Then, a range sum problem can compute the number of animals (RS) within a reachable range (R) from the supply deployment location. This can help decide the volume of supplies to be deployed.

* Corresponding author.

** Part of this work is done when Jianzhong Qi is visiting Northwestern University, which is supported by the Eshbach Visiting Scholar Fellowship.

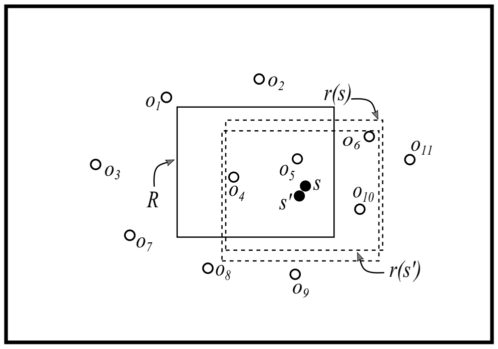


Fig. 1. A range sum problem

Figure 1 illustrates the range sum problem, where $o_1, o_2, ..., o_{11}$ are the data objects (mobile users or animals), and the solid rectangle at the middle represents the given range R. There are two objects o_4 and o_5 inside R. Thus, the RS is 2, assuming that each data object has a weight of 1.

Recent RS studies [3], [4], [5], [6], [7], [8] have focused on the *maximum range sum* (MaxRS) query. The MaxRS query finds a location to place the centroid of rectangle R such that its RS is maximized. Allocating resources (e.g., setting up a base station or deploying supplies) to this location will impact the most data objects (e.g., mobile users or animals). In Fig. 1, placing the centroid at the black point s results in a dashed rectangle $r(s)$ that encloses four data objects o_4, o_5, o_6, and o_{10}. No other rectangle of the same size can enclose more data objects. Thus, s is an answer to the MaxRS query.

Knowing only the location with the maximum RS, however, may be insufficient. Such a location may not be suitable for setting up a base station or deploying animal supplies due to various constraints, e.g., being a private property or in the middle of a river. Further, decision making processes such as resource allocations involve multiple factors rather than simply maximizing the RS. Decision makers may want to know the RS of different locations to make fully informed decisions.

To provide decision makers an overview of the RS at every location in the data space, we propose to compute a *range sum heat map* (RSHM). An RSHM associates each point in the data space with its RS.

Definition 1 (Range sum heat map): Given a set of points O where each point $o_i \in O$ has a weight represented by a real value $o_i.w$, a rectangular range R, and a two-dimensional Euclidean data space S, the *range sum heat map* associates each point $s \in S$ with its range sum, denoted by $s.rs$:

$$s.rs = \sum_{o_i \in O, o_i \in r(s)} o_i.w$$

Here, $r(s)$ represents the rectangle centered at s with the same shape as the given range R.

While there are infinite points in a Euclidean space and computing the RS for every point may seem difficult, a simple observation helps overcome this difficulty, i.e., the RS of nearby points are usually the same. In Fig. 1, the rectangles $r(s)$ and $r(s')$ of two nearby points s and s' both enclose data objects o_4, o_5, o_6, and o_{10}. Thus, s and s' have the same RS 4.

Based on this observation, we can divide the data space S into partitions, each of which is formed by points with the same RS. We only need to compute an RS for each partition. Computing an RSHM then becomes a problem of partitioning S. Figure 2 illustrates an RSHM created by partitioning the space with some dashed rectangles. The gray partition, for example, is formed by points with the same RS of 4. A range R centered at any point s in this partition (e.g., the solid rectangle $r(s)$) encloses four objects o_4, o_5, o_6, and o_{10}.

Each dashed rectangle used for the partitioning is centered at a data object o_i and has the same shape as R. Such a rectangle is called the *Minkowski rectangle* (MR) of o_i, denoted by mr_i. Any point s enclosed by mr_i can count o_i into its RS because a rectangle $r(s)$ centered at s must also enclose o_i. For example, point s in Fig. 2 is enclosed by mr_4, mr_5, mr_6, and mr_{10}. The four corresponding data objects are enclosed by $r(s)$.

Given an RSHM, a decision maker can easily examine the potential impact of allocating resources (e.g., setting up base stations or supply points) to different partitions. This allows allocating resources in a more flexible way to avoid unviable locations. The heat map can also help choose locations in multiple partitions that together have a large aggregate impact rather than a single location with the maximum RS.

We further consider data insertions and deletions for dynamic application scenarios. For example, mobile users may be active from time to time; and animal tracking sensors may be switched on and off to conserve battery power. To handle such scenarios, we propose to maintain an RSHM continuously over dynamically updating data points. We focus on data insertions and deletions. A location update of a data point can be handled by deleting the data point and re-inserting the data point with a new location. We call this problem the *continuous range sum heat map maintenance* (CRSHMM) problem.

Definition 2 (CRSHMM problem): The *continuous range sum heat map maintenance* problem computes, at every time point from the request being issued until deactivated, the range sum heat map for an Euclidean data space S over a set of data points O with dynamic insertions and deletions.

The CRSHMM problem offers a dynamic partitioning of the data space and visualization of the RS of different partitions. This can be used to support real-time resource allocation applications. For example, an online computer game server may compute an RSHM on its game map space using the players as the data points. This can guide the deployment of

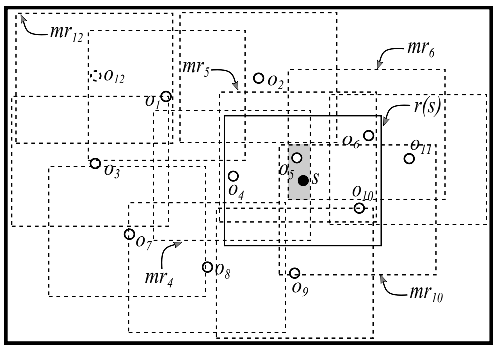


Fig. 2. A range sum heat map

gaming objects (e.g., non-player characters) to locations with large RS so that more players can interact with them. As the players may get online and offline frequently, maintaining an RSHM continuously is important for such applications.

Several studies consider RS computation over streaming data [4], [5], [8] or moving objects [7]. These studies focus on continuous MaxRS query maintenance. Branch and bound techniques are proposed to prune unpromising regions that cannot produce the maximum RS. Such techniques are inapplicable for the CRSHMM problem because the entire data space is of interest in this problem.

We propose an algorithm for efficient RSHM maintenance. The key idea of the algorithm is to only perform local updates to the partitions with data updates rather than recomputing the entire heat map. Take Fig. 2 as an example. If a new data point o_{12} is inserted, only those partitions that overlap with the MR of o_{12}, mr_{12}, will be affected and need to be updated. Similarly, if a data point, e.g., o_4, is deleted, only the range sums of the partitions enclosed by mr_4 need to be reduced by the weight of o_4. We build a spatial index over the partitions for fast retrieval of the partitions affected by a data update.

To summarize, this paper makes the following contributions:

1) We propose the problems of computing range sum heat maps and continuously maintaining such heat maps over data objects with dynamic insertions and deletions.
2) We adapt algorithms from related studies for efficient range sum heat map computation and maintenance.
3) We build a demo system to showcase the usefulness of the proposed problems and the effectiveness of the heat map computation and maintenance algorithms.

The rest of the paper is organized as follows. Section II discusses how to maintain an RSHM continuously. Section III details the demo system. Section IV concludes the paper.

II. Continuous Heat Map Maintenance

We start with computing range sum heat maps and then discuss how to maintain them with data updates.

A. Range Sum Heat Map Computation

To compute an RSHM, we need to compute all the partitions formed by the intersections of the MRs of the data points. A basic observation here is that the partitions must have a rectangular shape, as intersections of rectangles are still rectangles. The gray rectangular partition in Fig. 2, for example, is the intersection of four MRs mr_4, mr_5, mr_6, and mr_{10}.

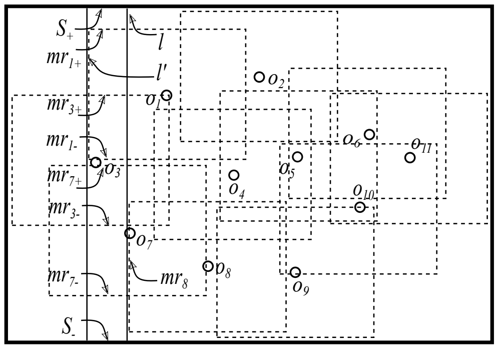


Fig. 3. Range sum heat map computation

Based on this observation and previous studies [5], [7], [8], [9] on range sum problems, we impose an order over the checking of MR intersections using a *plane-sweep* algorithm. The aim is to reduce the number of possible intersections to be checked. As shown in Fig. 3, the key idea of the algorithm is to use a vertical line l to sweep through the data space. As l moves, we store the MRs reached by it in a set M. Every time the line reaches (or leaves) an MR, we compute the intersections among the MRs stored in M. In the figure, l sweeps from the left to the right. It now reaches mr_8. The set M contains mr_1, mr_3, and mr_7. To compute the MR intersections, we sort the y-coordinates of the MRs in M. This yields a list of coordinates $L = \langle S_-, mr_{7-}, mr_{3-}, mr_{7+}, mr_{1-}, mr_{3+}, mr_{1+}, S_+\rangle$. Here, S_- (mr_{i-}) and S_+ (mr_{i+}) represent the lower and upper y-coordinates of the data space S (mr_i), respectively. Every pair of adjacent y-coordinates in L (e.g., mr_{7-} and mr_{3-}) form the lower and upper y-coordinates of a partition. The current x-coordinate of l serves as the upper x-coordinate of the partition, while the x-coordinate of l when it reached (or left) the preceding MR (mr_1) serves as the lower x-coordinate (cf. l'). A scan over L generates all partitions at l. The RS of the partitions are computed during the scan. Starting from the first pair of adjacent y-coordinates, we add (subtract) the weight of the corresponding object o_i to (from) the RS every time we reach the lower (upper) y-coordinate of an MR mr_i. For example, when we reach mr_{7-}, we add the weight of o_7 to the RS. Any partition created afterwards includes this weight in its RS until mr_{7+} is reached.

After generating the partitions, we also add (remove) the MR just reached (left) by l to (from) M, e.g., mr_8 in Fig. 3.

We note that a similar algorithm has been proposed for computing regions with the same number of *reverse nearest neighbors* [10]. The RSHM algorithm described above can be seen as a simplified version of that algorithm.

B. Heat Map Maintenance

When there are data updates, we observe that not the entire heat map needs recomputation. A data point o_i inserted into or removed from the data space only impacts the RS of the partitions overlapped by the MR mr_i of o_i.

We perform local updates to the RSHM instead of full recomputations. We build a spatial index over the partitions to help identify the partitions impacted by a data update. We use the R-tree, although other spatial indices may be used as well. At each time point, we group the MRs of the data points that have been updated together and join them with the R-tree T over the partitions. Once the partitions overlapping these MRs are identified, we update them as follows. For the partitions overlapping MRs of inserted points, we run the RSHM computation algorithm described above over the partitions together with the MRs to gain new partitions. For the partitions overlapping MRs of deleted points, we simply subtract the weights of the corresponding points from their RS.

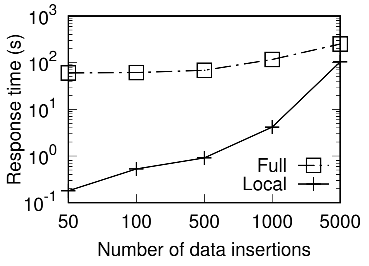


Fig. 4. Effect of the number of data insertions

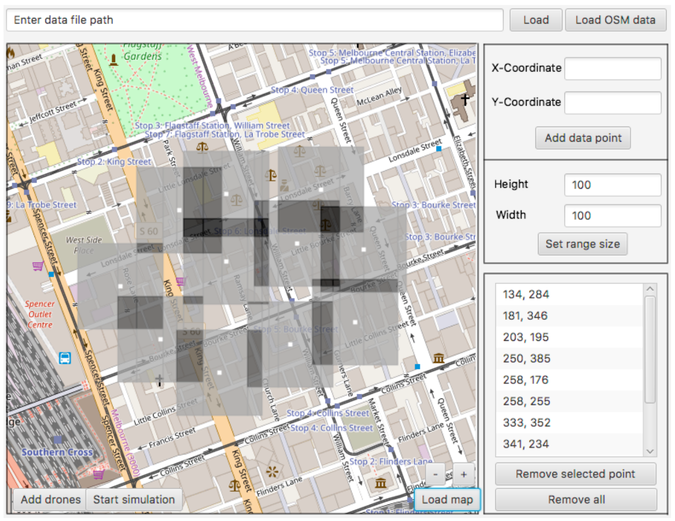
We run experiments to compare the efficiency of the proposed local update algorithm with a straightforward full recomputation algorithm. We implement both algorithms in-memory using Java. The experiments are run on a 64-bit Window 10 machine with an Intel Core i7 2.60Hz CPU and 8GB memory. The R-tree on the partitions has a fan-out of 50.

We generate 5,000 data points randomly distributed in a 1,000 × 1,000 space. We use a 20 × 20 range R. We first compute a heat map over these 5,000 data points. This takes 59.84 seconds. To test the efficiency of the heat map update algorithms, we vary the number of data points to be inserted from 50 to 5,000. These data points also follow a random distribution. We report the response time of the two algorithms in Fig. 4. We see that the proposed local update algorithm (denoted by *local*) outperforms the full recomputation algorithm (denoted by *full*) for the different numbers of data insertions tested. When the number of insertions is small, e.g., 50, the proposed algorithm is more than two orders of magnitude faster than the baseline algorithm (note the logarithmic scale). We also conduct experiments to vary the number of data deletions, the number of initial data points, the data distribution, and the range size. The local update algorithm outperforms the full recomputation algorithm consistently in these experiments. We omit the figures due to the page limit.

III. Demonstration

Next, we build a demo system to demonstrate the usefulness of RSHM and the effectiveness of the proposed RSHM maintenance algorithm. As shown in Fig. 5, the demo system consists of a canvas and a control panel alongside.

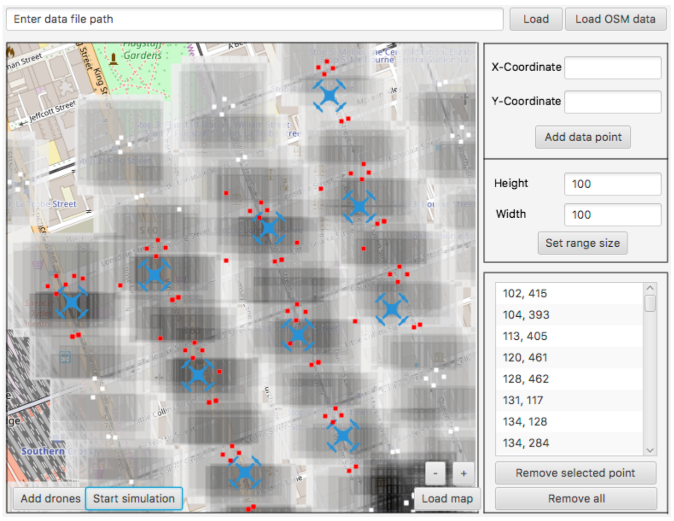
The canvas is used to draw the heat map, where the data space is partitioned by the MRs of the data points. Each partition has a different gray scale color to denote its RS. We use a weight of 1 for each data point to keep the demo system succinct, but it is straightforward to add varying weights. Data points can be added by clicking on the canvas or removed by right-clicking the points. When a data point is added



Height 100
Width 100

134, 284
181, 346
203, 195
250, 385
258, 176
258, 255
333, 352
341, 234

Fig. 5. The demo system



Height 100
Width 100

102, 415
104, 393
113, 405
120, 461
128, 462
131, 117
134, 128
134, 284

Fig. 6. A traffic surveillance scenario

or removed, the proposed local update algorithm is called to update the heat map. In this figure, we use the map of the city of Melbourne as the background. Maps from other cities or a blank background can be used as well. At the top of the demo system, we allow bulk-loading data points that we generate and data points collected from OpenStreetMap (https://www.openstreetmap.org) data dumps.

The control panel supports an alternative way of adding or removing data points based on coordinates. It also supports setting the size of the query range R.

To demonstrate the usefulness an RSHM, we simulate a scenario where a number of drones are to be deployed for traffic surveillance at intersections. We load intersection points from the OpenStreetMap data. We then add drones and set their flying trajectories in the canvas. To maximize the number of traffic intersections that are monitored by the drones, we can set the trajectories of the drones such that they pass as many darker regions (i.e., partitions with larger RS) as possible. In Fig. 6, we simulate 10 drones flying in the canvas, each can monitor a range of 100×100. We set the trajectories of the drones to pass mainly the darker regions. As highlighted by the red dots, most of the intersections are under surveillance. Without the heat map, it is more difficult to design trajectories for the drones to cover most intersections at the same time.

IV. Conclusion

We studied continuous maintenance of range sum heat maps over dynamically updating data points. This problem provides a visualization of the range sums at different locations in the data space, which helps decision making processes such as resource allocations. We adapted algorithms from related studies for range sum heat map computation. We further proposed an algorithm that used local update for efficient continuous maintenance of range sum heat maps with data updates. Our experimental results showed that the proposed heat map maintenance algorithm consistently outperformed a baseline algorithm that recomputes the entire heat map. Our demo system showcased the usefulness of the range sum heat maps and the effectiveness of the proposed algorithm.

For future work, an even more dynamic version of the demo system enabling fast continuous updates could be built and accommodated with our algorithms and data structures based on ideas from [11]. Computing heat maps for objects with non-zero extends would also be an interesting direction to explore.

Acknowledgment

This work is supported by Australian Research Council (ARC) Future Fellowships Project FT120100832 and Discovery Projects DP130104587 and DP180102050, The University of Melbourne Early Career Researcher Grant (Project Number: 603049), NSF Grants III 1213038 and CNS 1646107, ONR Grant N00014-14-10215, and HERE Grant 30046005.

References

[1] Y. Tao and D. Papadias, “Historical spatio-temporal aggregation,” *ACM Trans. Inf. Syst.*, vol. 23, no. 1, pp. 61–102, 2005.

[2] H.-J. Cho and C.-W. Chung, “Indexing range sum queries in spatio-temporal databases,” *Inf. Softw. Technol.*, vol. 49, no. 4, pp. 324–331, 2007.

[3] D.-W. Choi, C.-W. Chung, and Y. Tao, “Maximizing range sum in external memory,” *ACM Trans. Database Syst.*, vol. 39, no. 3, pp. 21:1–21:44, 2014.

[4] D. Amagata and T. Hara, “Monitoring maxrs in spatial data streams,” in *EDBT*, 2016, pp. 317–328.

[5] D. Amagata and T. Hara, “A general framework for maxrs and maxcrs monitoring in spatial data streams,” *ACM Trans. Spatial Algorithms Syst.*, vol. 3, no. 1, pp. 1:1–1:34, 2017.

[6] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, X. Cheng, and P. Chen, “Rotating maxrs queries,” *Inf. Sci.*, vol. 305, no. C, pp. 110–129, 2015.

[7] M. M. Hussain, K. A. Islam, G. Trajcevski, and M. E. Ali, “Towards efficient maintenance of continuous maxrs query for trajectories,” in *EDBT*, 2017, pp. 402–413.

[8] M. I. Mostafiz, S. M. F. Mahmud, M. M. Hussain, M. E. Ali, and G. Trajcevski, “Class-based conditional maxrs query in spatial data streams,” in *SSDBM*, 2017, pp. 13:1–13:12.

[9] “A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids,” *Computers and Mathematics with Applications*, vol. 29, no. 8, pp. 45–61, 1995.

[10] Y. Sun, R. Zhang, A. Y. Xue, J. Qi, and X. Du, “Reverse nearest neighbor heat maps: A tool for influence exploration,” in *ICDE*, 2016, pp. 966–977.

[11] E. Tanin, R. Beigel, and B. Shneiderman, “Design and evaluation of incremental data structures and algorithms for dynamic query interfaces,” in *IEEE Symposium on Information Visualization*, 1997, pp. 81–86.