Maximum Range-Sum for Dynamically Occurring Objects with Decaying Weights

Ashraf Tahmasbi(✉) and Goce Trajcevski

Iowa State University, Ames, IA 50011, USA
{tahmasbi,gocet25}@iastate.edu

Abstract. This work addresses a novel variant of the Maximum Range-Sum (MaxRS) query for settings in which spatial point objects occur dynamically and, upon occurrence, their significance (i.e., weight) decays over time. The objective of the original MaxRS query is to find a location to place (the centroid of) a fixed-size spatial rectangle so that the sum of the weights of the point objects in its interior is maximized. The unique aspect of the problem studied in this paper, which we call DDW-MaxRS (Dynamic and Decaying Weights MaxRS), is that the placement of its solution can vary over time due to the joint impact of the arrival of new objects and the change of the corresponding weights of the existing objects over time. To improve the efficiency of the DDW-MaxRS problem processing, we propose a memory-efficient approximate algorithmic solution that will naturally infuse uncertainty in the answer. We formally analyze the error bounds' properties and provide experimental results to quantify the effectiveness of the proposed approach.

Keywords: Maximum range-sum · Dynamic objects · Approximation

1 Introduction

The advances in GPS-equipped mobile devices and networking technologies have enabled generation of large volumes of location-aware data [7,20]. Often, in addition to location and time, such data includes various semantic (numerical and/or categorical) attributes, which are of interest in many applications relying on location-based services (LBS), such as social networks, emergency response, recommendations, etc. [18]. These settings, in turn, have brought about novel problems and challenges in query processing and analytics, the efficient management of which requires methodologies that extend the traditional spatial/Spatio-temporal and Moving Objects Database (MOD) approaches [8,25].

One such problem (re)addressed in the recent years is the Maximum Range-Sum (MaxRS) query, also known as the Maximum-enclosing Rectangle Problem [26], described as follows: given a set of (weighted) spatial point-objects O and a rectangle R with fixed dimensions $(a \times b)$, determine the location for placing R such that the sum (of weights) of the objects in R's interior is maximized.

Research supported by the NSF SWIFT grant 2030249.

S. Chiusano et al. (Eds.): ADBIS 2022, LNCS 13389, pp. 238–252, 2022.
https://doi.org/10.1007/978-3-031-15740-0_18

Figure 1 shows the positions of a set of input points and their respective weights. The shaded region indicates the position of the query rectangle such that the sum of the weights of the (red-colored) objects in its interior is maximum among all the other possible placements in the horizontal plane.

What motivates this work is that in certain domains, in addition to the arrival/occurrence of new points, one may face a situation in which the weights of the previously inserted objects need not be constant over time (e.g., it may decay). For example, when monitoring a CO concentration through participatory sensing [3], mobile citizens periodically report the value of the measurement from their current location, based on incentives [11]. One of the objectives of managing carbon footprint is to avoid uneven distribution in urban environments [14,19]. A traffic management system may want to enforce reduced driving through an area of a given size (limited by the constraint not to cause too severe congestion) for some time. As time evolves, when re-evaluating the critical area to reduce traffic, one must consider that the value of ppm (particles per million) measured in a previous location of the mobile volunteer may have declined over time.

A complimentary scenario comes from burglary crime management. To determine the placement of a fixed-size region that will cover a maximum number of ongoing burglaries [17] – upon reporting a new one, one needs to consider that the typical "stay-time" of burglars for each previously reported one in a location is between 8–12 min [13]. Similarly, when estimating the (placement of a) fixed size region in which there is the highest likelihood of getting infected by a virus due to the concentration of infected persons, one needs to consider that the capacity of transmitting the infection decays over time [21].

While at their core, such problems resemble the settings of the MaxRS problem, they also accentuate the joint impact of two factors: (i) dynamic appearances of objects at given locations, with varying weight/importance (e.g., the severity of symptoms); and (ii) variability over time of the objects' weights.

None of the available variants of the MaxRS problem can be straightforwardly applied to address the settings described above. Towards that, the main contributions of this work are threefold: (i) We take a first step towards for-

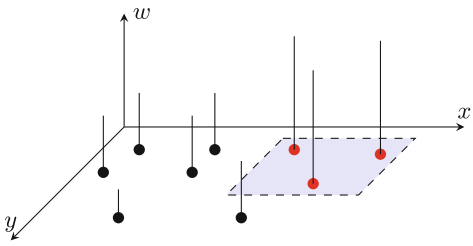

Fig. 1. Location of MaxRS at time t; weights of each object are shown as its third dimension over the z axis.

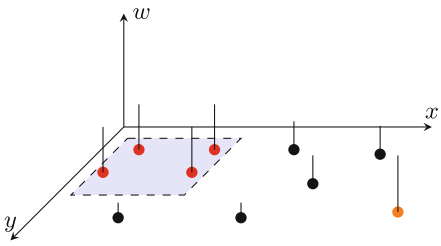

Fig. 2. MaxRS at time $t+1$; decayed weights changed the answer and a new WD object arrived (shown in orange). (Color figure online)

malizing a novel *Dynamic and Decaying Weights MaxRS* (DDW-MaxRS) query; (ii) We identify and formally prove properties that ensure desired bounds for a novel approximate solution and introduce a corresponding algorithm; and (iii) We present experiments showing the effectiveness of the proposed approach.

We introduce the notation and formalize the problem in Sect. 2, followed by the algorithmic solutions in Sect. 3. We discuss generalizations of the parameters in Sect. 4 and present experimental observations in Sect. 5. Related literature is presented in Sect.6 and the concluding remarks in Sect. 7.

2 Preliminaries

We consider a time-varying set of objects $O_{n_t}(t) = \{o_1, o_2, \ldots, o_{n_t}\}$, each of which is associated with a location in 2D Euclidean space, within a suitable coordinate system. Each object o_i is a triplet $((x_i, y_i), w_i(t), t_i)$, where i is its unique identifier, t_i is the time step in which this object arrived, (x_i, y_i) represents its location, and $w_i(t)$ is a scalar indicating its (relative) importance – e.g., the capability for spreading infection over time – defined as:

$$w_i(t) = \begin{cases} w_i \times \lambda_i^{(t-t_i)}, & \text{if} \quad t \geq t_i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where w_i indicates its initial importance/weight, and $\lambda_i < 1$ captures the decay over time – i.e., the diminishing of the importance of a particular object. In the rest of this paper, we assume that at each time step t, at most, one new object o_i is generated in the regions of interest. For brevity, we will refer to such objects as WD (weighted and decaying) objects in the rest of this paper. For a given axis-parallel rectangle R, we use $C(l_R, R)$ to denote the planar region covered by R when its center is placed in location l_R.

The Dynamic and Decaying Weights variant of the MaxRS over a time interval $\Delta(t) = [t_s, t_e]$ (DDW-MaxRS$((O_{n_s}(t)), R, (t_s, t_e))$) is defined as follows:

Definition 1. *Given a (starting) set $O_{n_s}(t_s)$ of WD objects $O_{n_s}(t) = \{o_1, o_2, ..., o_{n_s}\}$, and a query rectangle R, the answer to* DDW-MaxRS$((O_{n_s}(t)), R, (t_s, t_e))$*, denoted $A_{\text{DDW-MaxRS}}$ $(O_{n_s}(t), R, (t_s, t_e))$ is a sequence of locations $L_R = \{l_R(t_s), \ldots, l_R(t_e)\}$ for placing the centroid of R, such that at each $t \in [t_s, t_e]$, $\sum_{\{o_i \in (O_{n_t}(t) \cap C(l_R, R))\}} w_i(t)$ is maximal.*

We are interested in two key aspects: (1) trade-offs that arise when sacrificing the accuracy of the answer for the benefit of the efficiency of the processing algorithm; and (2) avoiding a complete re-computation of the $A_{\text{DDW-MaxRS}}$ $(O_{n_s}(t), R, (t_s, t_e))$ at every time instant (i.e., naively invoking the traditional Max-RS algorithm for each $O_{n_t}(t)$). For (1), we consider a variant similar to [22], introducing a factor of ε to quantify the error compared with the exact answer. We have the following definition (updating the corresponding definition of $(1-\varepsilon)$-Approximate MaxRS from [22]) for an approximate answer at a time instant t:

Definition 2. *Given a set $O_n(t)$ of n WD objects $O_n(t) = \{o_1, o_2, ..., o_n\}$, the answer to $(1-\varepsilon)$ Approximate DDW-MaxRS query at any time t, denoted $A_{\varepsilon\ \text{DDW-MaxRS}}(O_{n_t}(t), R)$, is a position $l_R(t)$ for placing the center of R such that the covered wight of R is a $(1-\varepsilon)$-approximate answer, i.e.,*

$$(1-\varepsilon) \times m^* \leq m \leq m^* \tag{2}$$

where $m = \sum_{\{o_i \in (O_n(t) \cap C(l_R, R))\}} w_i(t)$ is the weight returned by $A_{\varepsilon\ \text{DDW-MaxRS}}$ $(O_n(t), R)$, and m^* is the weight returned by $A_{\text{DDW-MaxRS}}(O_n(t), R)$.

However, in our case, answers may vary over time. For illustration, consider Fig. 2, showing a future time instant of the scenario in Fig. 1. It shows the change of the (location of the) answer due to the different decays of the weights over time. In addition, it shows an arrival of a new object (bottom-right) which, however, does not affect the answer.

3 Approximate Solution to *DDW-MaxRS*

We note that a naïve way to calculate the answer to DDW-MaxRS would be to invoke the traditional MaxRS solution (cf. [6]) at each time step, incorporating a new WD object, and weight modifications of the older object due to the decay over time. The time complexity of the exact solution, in this case is $O(n_t \log n_t)$ for each time instant t, where n_t is the (progressively increasing) number of WD objects at time t. To avoid such re-computations every time instant, one can rely on the approach to maintain the MaxRS in dynamic settings [2].

In the rest of this Section, we focus on a specific case of the DDW-MaxRS problem, where we assume a fixed decay – i.e., $\lambda_i = \lambda$ for all objects/points (we will later relax this assumption in Sect. 4).

To explain the intuition behind the approximate solution, let S denote the 2D space of all the possible locations for all the objects. Assuming a query rectangle R of size $[a \times b]$, we partition this space into cells of the same size $[a/j \times b/j]$ for some value j. Without loss of generality (i.e., with small "boundary zones"), we assume that S is fully covered by a 2D array of a whole number of cells. Upon arrival, each new object o_i, based on its location, is mapped to a cell. For each cell $c_{u,v}$, we store a list of points mapped to it. Given this discretization of S, upon an arrival of a new object o_m at some time instant t_m, we introduce its *extended dual rectangle* $R^e_{o_m}$ of a size $[(2j+1)a/j \times (2j+1)b/j]$ cells, centered in the center of the cell to which o_m belongs.

As we will formally demonstrate below, $R^e_{o_m}$ enables a more efficient calculation of the $A_{\varepsilon\ \text{DDW-MaxRS}}$ $(O_{n_M}(t_m), R)$ based on the solution at the previous time instant $t_m - 1$. The rationale is that if there is a change in the solution since $t_m - 1$, it must be due to o_m and a subset of the older points located in the cells that are intersecting $R^e_{o_m}$.

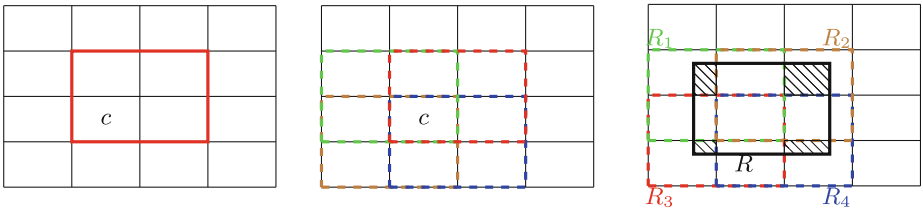

Fig. 3. For $j = 2$: (a) extended cell $ExtCell(c)$ (b) All extended cells containing c, (c) Naive approach provides at least a $\frac{1}{4}$-approximate solution to DDW-MaxRS.

Algorithm 1: Grid-based Approximate Solution

input: ExtCList is the sorted list of ExtCells based on their weights, p^* the solution to A_{ε} DDW-MaxRS$(O_n(t), R)$ with weight w^*

`// Target rectangle` R `is of size` $a \times b$

if *new arrival* o_i **then**
  Find the cell c that o_i maps into
  $CellSet(c) \leftarrow CellSet(c) \cup o_i$
  $W(c,t) \leftarrow W(c,t) + w_i(t)$
  for $ExtCell(d)$ *containing* c **do** `// There are` j^2 `of such extended cells`
    $W(ExtCell(d),t) \leftarrow W(ExtCell(d),t) + w_i(t)$
    Update the position of $ExtCell(d)$ in ExtCList to keep it sorted
      `// this step takes` $O(\log N)$`, where` N `is the total number of cells`
  end
  $p^* \leftarrow$ center of ExtCList$[-1]$ `// ExtCList`$[-1]$ `is the ExtCell with the maximum weight`
  $w^* \leftarrow W(ExtCellList[-1], t)$
end
else
  $w^* \leftarrow \lambda \times w^*$
end

Recall that we aim for an approximate solution(s) which, at each time instant, provides a faster calculation of the answer(s) at the price of sacrificing accuracy in terms of the covered weight. We quantify this error by ε as defined in Definition 2.

For each cell c, we store a list of points mapped to it (OList(c)). At any time t, the weight of a cell c is computed as: $W(c,t) = \sum_{\{o_i \in \text{OList}(c)\}} w_i(t)$. We also define an extended cell ExtCell(c) as the set of cells (CellSet(c)) in the grid of $j \times j$ cells with cell c as its most left-bottom cell (shown in Fig. 3a). The weight of an extended cell ExtCell(c) is computed as: $W(ExtCell(c),t) = \sum_{\{cell \in CellSet(c)\}} W(cell,t)$.

We propose an approximate solutions based on the idea of diving the space S into cells, and the pseudo code is shown in Algorithm 1.

3.1 Properties of the Approximate Solution

To return an approximate solution to the DDW-MaxRS problem, we do as follows: (i) At the initial time t_0, we keep a list of extended cells and order them based on their weights (we will refer to this list as ExtCList)). We then return the extended cell with the highest weight as the approximate solution. (ii) Let ExtCell(c^*) be our approximate solution to DDW-MaxRS at time $t-1$ (for any $t > t_0$). We will not change the approximate solution if no new point arrives at time step t. We only update its weight. However, if a new point arrives, we first find the cell c this point maps into it. Note that there are j^2 extended cells containing c (shown in Fig. 3b for the choice of $j = 2$). We update the weight of c and the associated extended cells and their position in the ExtCList so that our list remains sorted. We then return the extended cell with the maximum weight as the solution to our DDW-MaxRS. The critical thing to remember is that the relative order of all the other extended cells remains the same.

Lemma 1. *At any time $t > t_0$, upon arrival of a new point o, the relative order of extended cells, that o does not map into, remains intact.*

The proof is rather straightforward, based on the fact that the weight of each extended cell that the new point is not mapped to decay by a factor of $\lambda < 1$. This algorithm provides *at least* $\frac{1}{4}$-approximate solution (cf. Lemma 2).

Lemma 2. *At any time t, the grid-based algorithm returns at least $\frac{1}{4}$-approximate solution to the DDW-MaxRS problem.*

Proof. For simplicity, we discuss $j = 2$, but the proof for larger values of j is similar. Let m be the weight of the extended cell returned by the algorithm at time t. Also, let the exact solution to the DDW-MaxRS problem, at time t, be as shown in Fig. 3c, which means it does not fall within any of our defined extended cells but rather overlaps with multiple of them (R_1, R_2, R_3, and R_4).

The proof follows the fact that the weight of points covered in the intersection of the exact solution R with each of them is bounded by m, and hence, the total weight covered by R cannot be more than $4m$. Note that this is a *tight bound* □.

Intuitively, larger values for j imply a finer grid, and one may expect that it will improve the accuracy. However, having a very large j could result in cells containing only 1 (or 0) point, and there will be no benefit in dividing the space into cells. There is a trade-off in the choice of j between the accuracy of results and the computational efficiency. We study the effect of this parameter on the results in Sect. 5.

3.2 Memory-Efficient Approximate Solution

When mapping WD objects to grids, not all cells will include a point. A significant number of cells will be either empty or receive objects infrequently. It does not makes sense to store such objects or keep track of these sparse cells. This sub-section introduces an approach to clear sparse cells without hurting the accuracy. To this end, let us formally define a sparse cell:

Definition 3. *A cell c is called θ-sparse at time t if:*

$$W(c,t) \leq \theta \tag{3}$$

According to Definition 3, we call a cell θ-sparse at time t if its weight is less than θ, where $\theta > 0$ is a threshold we set. Later, We will discuss how to choose this threshold. Lemma 3 investigates lower and upper bounds on the sum of all WD objects' weights present in the system at any time t:

Lemma 3. *Let t_l be the last time step in which a new point arrived. At any time step $t \geq t_l$, we have:*

$$(n_0+1)\lambda^t < \sum_{\{o_i \in O_{n_t}\}} w_i(t) < (n_0-1)\lambda^t + \frac{1}{1-\lambda} \tag{4}$$

where n_0 is the number of points in the system at time $t = 0$.

Proof. The proof is straightforward: given n_0 WD objects were present in the system at time t_0, the total weight is at least $n_0\lambda^t + \lambda^{t-t_l}$. On the other hand, in the worse case, we had one new WD object added at each time step from $t = 0$ to $t = t_l$, and the total weight of those objects is bounded by $\frac{1}{1-\lambda}$. □

In the rest of this section, we assume $n_0 = 1$ for simplicity, but the results are generalizable to $n_0 > 1$.

Let an algorithm clear all the θ-sparse cells (for some predetermined θ). One question that arises is how does it affect our grid? How many nonempty cells will remain after clearing such cells? Lemma 4 answers this question:

Lemma 4. *At any time t, the number of nonempty cells after deleting the θ-sparse cells will never exceed $L = \log_\lambda \theta(1-\lambda)$.*

Proof. The proof is similar to the proof of Theorem 4.4 in [23]. Let t_c be the last time instant in which cell c received a new point. Therefore, we have:

$$W(c,t) = \lambda^{t-t_c} \times W(c,t_c) \tag{5}$$

Consider the case when $t - t_c > L = \log_\lambda \theta(1-\lambda)$, then we have:

$$W(c,t) \overset{(i)}{=} \lambda^{t-t_c} W(c,t_c) < \lambda^L W(c,t_c) \overset{(ii)}{<} \frac{\lambda^L}{1-\lambda} = \frac{\theta(1-\lambda)}{1-\lambda} = \theta \tag{6}$$

Where (i) holds because of (5) and (ii) is true because, according to Lemma 3, the sum of all weights at time t is less than $1/(1-\lambda)$ (for $n_0 = 1$); therefore, the weight of each cell is also less than this upper bound. 6 concludes that any cell c that has not received any points within the last L time steps will be θ-sparse and erased at time t. Therefore, the number of non-empty cells is at most L. □

Our proposed memory-efficient solution is quite similar to Algorithm 1 with the main difference that θ-sparse cells (and consequently empty ExtCells) will be removed after line 4. As a consequence of this change, we need to address two fundamental questions:

Q1: *For any cell c, how much does the new weight differ from the actual weight of c if it was never cleared before?* Lemma 5 answers this question.

Lemma 5. *Suppose the last time a cell c was cleared because of being θ-sparse was t_d. If at current time t the weight of c is $\hat{W}(c,t)$, then we have:*

$$W(c,t) \leq \hat{W}(c,t) + (\frac{\theta}{1-\lambda^L})\lambda^{t-t_d} \tag{7}$$

where $L = \log_\lambda \theta(1-\lambda)$ and $W(c,t)$ is the actual weight of c if it was never deleted before.

Proof. The proof follows a similar approach as the proof of Theorem 4.3 in [23]. Suppose cell c has been previously deleted at time steps $t_1, t_2, ..., t_d$. Therefore, we knew that: $\hat{W}(c,t_i) \leq \theta$ for $i = 1, 2, ..., d$. Thus, if we choose θ to be a constant (not a function of t_is) and if we never cleared all these previous weights, the actual weight would be:

$$W(c,t) - \hat{W}(c,t) = \sum_{i=1}^{d} \hat{W}(c,t_i)\lambda^{t-t_i} \leq \sum_{i=1}^{d} \theta \times \lambda^{t-t_i} = \theta\lambda^{t-t_d} \times \sum_{i=1}^{d} \theta\lambda^{t_d-t_i}$$

On the other hand, according to Lemma 4, for each $i = 2, ..., d$, we have $t_i - t_{i-1} \geq L + 1$, where $L = \log_\lambda \theta(1-\lambda)$. Therefore $W(c,t) - \hat{W}(c,t) \leq \theta\lambda^{t-t_d}\left(\frac{1}{1-\lambda^L}\right)$. □

Q2: *How does the error in the weights of cells (as the result of clearing sparse cells) affect our approximate solution ?* Lemma 6 addresses this question.

Lemma 6. *At any time t, the proposed memory-efficient approximate solution with the choice of $\theta \leq \frac{1}{(1-\lambda)}\left(1-\frac{\lambda}{\alpha}\right)$, returns at least $\frac{1}{4(1+\alpha)}$-approximate solution to a DDW-MaxRS query, where $\alpha > \lambda$.*

Proof. Let c denote the cell with the maximum weight at time t. According to Lemma 5, we have $W(c,t) \leq \hat{W}(c,t) + \frac{\theta\lambda}{1-\lambda^L}$, where $L = \log_\lambda \theta(1-\lambda)$. Now, similar to the proof of Lemma 2, we can argue that the weight covered by R (the true answer of DDW-MaxRS at time t) is bounded as follows:

$$W(R,t) \leq 4 \times W(c,t) \leq 4 \times \left(\hat{W}(c,t) + \frac{\theta}{1-\lambda^L}\right) \leq 4 \times \left(1 + \frac{\lambda}{1-\lambda^L}\right)\hat{W}(c,t)$$

For the choice of $\theta \leq \frac{1}{(1-\lambda)}\left(1-\frac{\lambda}{\alpha}\right)$, we have $\lambda^L = \theta(1-\lambda) \leq (1-\frac{\lambda}{\alpha})$. Therefore, $(1 + \frac{\lambda}{1-\lambda^L}) \leq (1+\alpha)$. □

Upon arrival of a new point, the time complexity of the update is $O(j^2 \log L)$, where $L = \log_\lambda \theta(1-\lambda)$.

4 Generalization

So far, we have used certain assumptions on the settings of the DDW problem – i.e., having an equal initial weight for all objects, the same decaying factor, and a specific exponential decaying function. This section discusses how these assumptions can be relaxed and the consequences, i.e., what needs to be changed in the proposed solutions to make them suitable for more generalized variant(s).

Generalization to Different Initial Weights: The presented results can be generalized to cases where initial weights are of an interval value $[w_{\min}, w_{\max}]$ (i.e., $w_{\min} \leq w_i \leq w_{\max}$). Lemma 1 and 2 are directly applicable to such WD objects . Consequently, the approximate solution presented in Algorithm 1 can be applied and used for this generalized version. In addition, if we define $\theta' = w_{\max} \times \theta$, Lemmas 4, 5, and 6 will hold for this generalized version (i.e., by replacing θ with θ'). Thus, the proposed memory-efficient approximate solution can also be used to answer this particular generalized variant.

Generalization to Different Decaying Factors: If the initial weights for all WD objects are equal (i.e., $w_i = 1$), but their decaying factor is different ($\lambda_{\min} \leq \lambda_i \leq \lambda_{\max}$), Lemma 1 does *not* hold anymore, and the relative order of cells changes over time. One possible modification to Algorithm 1 is to check all the non-empty cells upon arrival of a new object and find the one with maximum weight. This would result in $O(m_t)$ time complexity at each time step t, where m_t is the number of the non-empty extended cells at time t ($m_t \leq j^2 \times n_t$). A complimentary observation is that one could ("naïvely") consider an upper bound, say, $\lambda_{\max}$ in the statements of the corresponding Lemmas. This would retain the properties in terms of the respective inequalities (possibly affecting the discrepancy between the values on the left-hand sides and right-hand sides). We note that while we provide experimental observations about the impact of λ in Sect. 5, a more detailed investigation is left for our future work.

Generalization to a Broader Class of Decaying Functions: Let us consider a class of functions F such that $\forall f \in F : f(t) \leq \epsilon(t) \times f(t-1)$, where $\forall t : \epsilon(t) \leq \epsilon(t-1) < 1$. If the decaying function of the weight of each WD object is from F, then Lemma 1 and 2 hold and apply directly to this more generalized case. As a result, Algorithm 1 can be used to provide an approximation solution to a DDW-MaxRS query. In addition, similar results can be proven by replacing λ with $\epsilon(1)$ in Lemmas 3–6.

5 Experimental Study

Since there are no existing solutions to the DDW-MaxRS problem, we implemented an adapted (i.e., no external memory) version of the grid-based solution from [6] as the baseline. We compared it against the Memory Efficient Approximate Solution (MEAS for short)[1] from Sect. 3.2. The comparison was done in

[1] Implementation and the datasets used are publicly available at https://github.com/Ashraf-T/DDW-MaxRS.

two aspects: (1) *Efficiency* (in terms of *speed up* compared to the baseline) and (2) *Accuracy* of the results in terms of $(1 - \epsilon)$.

Dataset: We use both synthetic and real-world datasets in our experiments. In synthetic datasets, objects are created using uniform and Gaussian distributions in a 2D space with a (relative) reference coordinate system . The real dataset we used is based on the check-ins in NYC collected for about ten months (from 12 April 2012 to 16 February 2013) [24]. We extracted the unique GPS coordinates (Latitude and Longitude) and mapped them to a Cartesian coordinate system. This transformation is done using pyproj, cartographic projections, and coordinate transformations library in Python[2] We converted the GPS coordinates from EPSG:4326 to a regional system EPSG:2263.

Table 1. Parameters

Parameter Name and Symbol	Possible Values	Default value
Object distribution	Uniform, Gaussian, Real	Uniform
No. of initial objects N_0	500, 1000, 5000, 10000, 50000	5000
Size of R	1.0×1.0, 1.5×1.5, 2.0×2.0, 2.5×2.5	1.0×1.0
Decay Factor (λ)	0.9, 0.8, 0.7, 0.6	0.9
Ratio of R to cells (j)	1, 2, 4, 8	2
Sparsity threshold (θ)	0.01, 0.05, 0.1, 0.2, 0.5	0.1

Parameters: The list of the studied parameters and their values is provided in Table 1. We note that the overall area of interest (i.e., the possible range for the locations of the objects) was bound by a rectangle of size (100×100). Unless otherwise indicated, when investigating the impact of a specific parameter, we keep the rest of them fixed to their respective default values, which are indicated in the rightmost column of Table 1. We scaled the x-y coordinates for the real dataset to be mapped to a rectangle of size (100×100). Also, for the experiments studying the impact of the number of initial objects, if available, we sampled as many data points uniformly at random from the dataset.

Environment: All the algorithms were implemented in Python 3.7. The experiments were executed on a machine with an Intel 2.2 GHz core i7 processor and 8 GB of RAM. We measured the median of the processing time and accuracy for the reported results over five runs.

Results: We now present the results in the two categories mentioned above.

(1) *Efficiency*: The results are presented in Fig. 4, which shows the values corresponding to the ratio of the processing time of $A_{\varepsilon\ \text{DDW-MaxRS}}$ relative to the processing time taken by the exact solution. The colored bars indicate the different datasets, and for each of them, the reported processing time is obtained

[2] cf. https://github.com/pyproj4/pyproj.

by averaging the processing time over 100-time steps through five iterations. We considered the impact of different parameters:
– N_0 - the initial number of WD objects present in the system: MEAS significantly reduces the processing time (about 45%) when there are a large number of initial objects. The processing time for the exact solution does not change much over time because its time complexity is a function of n_t (number of objects present at time t), which increases by (at most) one at each time step. Although the impact of the initial number in our settings is quite high, even in such cases, our proposed solution yields efficiency benefits. The processing time for the approximate solution also does not vary much over time, as its worst-case complexity is in order of $O(logN)$, where N is the number of extended cells in MEAS and does not change over time. Note that there is no bar corresponding to the real dataset with $N_0 = 50000$ since the number of unique x-y coordinates in the used dataset was not enough to sample these many records.
– R: As one would expect, the larger $|R|$, the faster the execution of the approximate algorithm. For the default values of the other parameters, in Fig. 4-b, we show the impact of the size of R on the speed up relative to executing the exact algorithm. We observe that similar trends hold in both approximate solutions, except for the results of the memory-efficient solution for the real dataset.

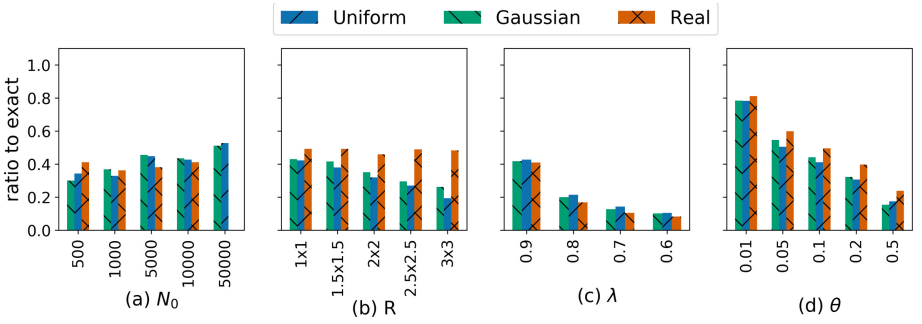

Fig. 4. Efficiency impact of (a) N_0, (b) $|R|$, (c) λ, and (d) θ

– λ: Larger decaying factor (λ) values diminish the weights of points by a greater factor after each time step. Therefore, MEAS will remove more cells as their total weights fall below the threshold and achieve a faster processing time than the exact solution and the first approximate solution in which the time complexity depends on the number of objects. Figure 4-c illustrates these results.
– θ: This parameter determines the cut-off line for the weight of sparse cells. MEAS clears more cells for the larger values of θ and achieves a higher speed up in its processing time. Figure 4-d shows the results for various choices of θ.

(2) *Accuracy*: This part presents the impact of the following four parameters: N_0, j, $|R|$ and θ. The results are shown in Fig. 5 only for the real-world dataset (the results for uniform and Gaussian distributions were similar).
– N_0: The effect of initial objects on the solution of DDW-MaxRS lasts longer for larger values of N_0. This means it may take a while before the solution of DDW-MaxRS changes due to the arrival of new objects. Figure 5-a shows the results of this experiment for MEAS. Again, no result is shown when $N_0 = 50K$ since there are not enough unique x-y coordinates in the real dataset to sample $50K$ of them.
– j - the ratio of the size of R to the size of the cells: Intuitively, a higher accuracy should be achieved for larger choices of j ; because the intersection of the optimal solution and the corner cells of an extended dual rectangle (illustrated by the hatch pattern in Fig. 3c) decrease by having smaller cells and consequently, the chance of objects being located in those areas also decreases. Figure 5-b confirms this expectation. Also worth mentioning that time complexity of the approximate solution is dominated by $log(N)$, where N is the number of extended cells, and the increase in the number of cells as the result of increasing j does not significantly affect the processing time.

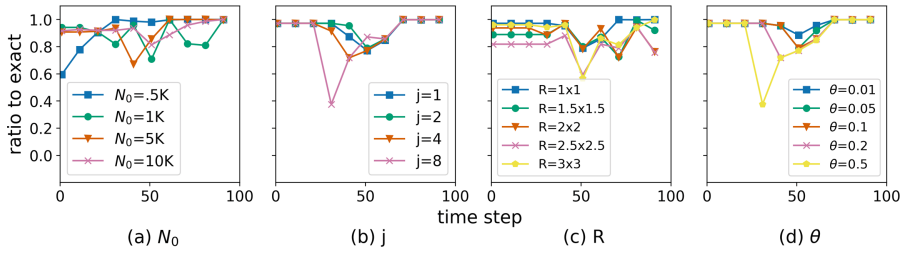

Fig. 5. Accuracy impact of (a) N_0, (b) j, (c) $|R|$, and (d) θ on MEAS for real-world dataset over time.

–R: One would expect that the larger size of the query rectangle decreases the accuracy of the approximate solution because there is a higher chance of having scenarios similar to the one shown in Fig. 3c. The reported results in Fig. 5-c support this expectation.

– θ: Choosing larger values of θ expectedly hurts the accuracy of the returned solution by MEAS. Setting larger values to θ will prune and erode more cells as sparse cells. Figure 5-d suggests that the memory-efficient solution performs pretty robust when θ is less than 0.5.

6 Related Work

The MaxRS problem has a long history, during which different variants have been addressed. It was first tackled by researchers in the computational geometry

community, where a technique that finds connected components and a maximum clique of an intersection graph of rectangles in the plane was proposed in [10]. A solution based on the plane sweep strategy was presented in [15], where the input points/objects were "dualized" into rectangles, centered at the locations of the input points and with dimensions equivalent to the query rectangle R. To obtain the regions with the highest number of intersecting (dual) rectangles, an interval tree was used, updated by a sweep line technique, with events at the input points (i.e., centroids of the respective dualized rectangles). The algorithm for possible locations for placing the (center of the) query rectangle yielding the maximal number of points in its interior had $O(n \log n)$ time complexity.

The next "era" of the MaxRS research was motivated by the observations that the scalability of the existing solutions may not be good for LBS application, and a scalable solution was proposed in [5]. An extension, considering imprecision vs. efficiency trade-off and providing an approximate solution, was presented in [22]. Dynamic settings, in which objects can be inserted and/or deleted, were considered in [2], based on an aggregate graph and yielding an efficient approximation. The work considered a sliding window-based model, in which the appearance of new objects implies the removal of old objects. Recently, the dynamic variant with a constraint on the minimal number of elements from different classes that must be inside the query rectangle was addressed in [9].

Other variants of the MaxRS problem have also been considered, such as constraining the locations of spatial points to the underlying road network [16, 26]; considering moving objects, and detecting the "trajectory" of the motion of the centroid of MaxRS; MaxRS [4] where rectangles do not need to be axes parallel.

A couple of recent works have addressed uncertain variants. In [1], an index structure was proposed for efficiently answering the MaxRS query in the d dimension, in which each point was associated with an existential probability. PMaxRS (cf. [12]) considered inherent location uncertainty of spatial objects, coupling candidates generation (pruning), and sampling-based approximation (refinement) to efficiently solve the problem.

While many of the above results have motivated our present work, there are some fundamental distinguishing aspects. Namely, we jointly consider the probabilistic nature of the dynamics of the arrival of new points along with the impact of the decay factor (i.e., the value of the weight) over time.

7 Conclusions and Future Work

We introduced a novel variant of the MaxRS query – the DDW-MaxRS, which, given a spatial region of interest, combines the consideration of the arrival of new objects and the decay of their weight over time. We took the first step toward providing an approximate algorithm $A_{\varepsilon\ \text{DDW-MaxRS}}(O_n(t), R)$ and formally analyzed error bounds properties compared with the exact solution. Our experiments demonstrated the impact of the various parameters and the benefits of the $A_{\varepsilon\ \text{DDW-MaxRS}}(O_n(t), R)$.

Our future work will focus on three main topics. Firstly, we will investigate data structures that can improve scalability. Secondly, will address the complementary settings of data streams – i.e., the arrival rate of the objects is fast, and the available memory is limited – and devise approximate algorithms based on effective sampling strategies. In parallel, we will explore the extension of our results to higher dimensions and the potential joint impact of different parameters (e.g., combining N_0 and $|R|$, with different distributions for N_0 and the newly arriving objects).

References

1. Agarwal, P.K., Kumar, N., Sintos, S., Suri, S.: Range-max queries on uncertain data. J. Comput. Syst. Sci. **94**, 118–134 (2018)
2. Amagata, D., Hara, T.: Monitoring MaxRS in spatial data streams. In: EDBT, pp. 317–328 (2016)
3. Boulos, M.N.K., et al.: Crowdsourcing, citizen sensing and sensor web technologies for public and environmental health surveillance and crisis management: trends, OGC standards and application examples. Int. J. Health Geograph. **10** (2011)
4. Chen, Z., Liu, Y., Wong, R.C., Xiong, J., Cheng, X., Chen, P.: Rotating MaxRS queries. Inf. Sci. **305** (2015)
5. Choi, D.W., Chung, C.W., Tao, Y.: A scalable algorithm for maximizing range sum in spatial databases. Proc. VLDB Endow. **5**(11), 1088–1099 (2012)
6. Choi, D., Chung, C., Tao, Y.: Maximizing range sum in external memory. ACM Trans. Database Syst. **39**(3), 21:1–21:44 (2014)
7. Eldawy, A., Mokbel, M.F.: The era of big spatial data: a survey. Found. Trends Databases **6**(3–4), 163–273 (2016)
8. Güting, R.H., Schneider, M.: Moving Objects Databases. Morgan Kaufmann, San Francisco (2005)
9. Hussain, M.M., Mostafiz, M.I., Mahmud, S.M.F., Trajcevski, G., Ali, M.E.: Conditional MaxRS query for evolving spatial data. Front. Big Data **3**, 20 (2020)
10. Imai, H., Asano, T.: Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. J. Algorithms **4**(4), 310–323 (1983)
11. Koutsopoulos, I.: Optimal incentive-driven design of participatory sensing systems. In: Proceedings of the IEEE INFOCOM (2013)
12. Liu, Q., Lian, X., Chen, L.: Probabilistic maximum range-sum queries on spatial database. In: Proceedings of the 27th ACM SIGSPATIAL (2019)
13. LLC, S.: Home Burglay awareness and prevention (2013). http://www.jsu.edu/police/docs/Schoolsafety.pdf
14. Mishra, A.: Data science: The key tool cities need to reduce carbon emissions (2020)
15. Nandy, S.C., Bhattacharya, B.B.: A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. Comput. Math. Appl. **29**(8), 45–61 (1995)
16. Phan, T., Jung, H., Youn, H.Y., Kim, U.: Efficient evaluation of maximizing range sum queries in a road network. IEICE Trans. Inf. Syst. **99-D**(5), 1326–1336 (2016)
17. Prieto Curiel, R., Collignon Delmar, S., Bishop, S.R.: Measuring the distribution of crime and its concentration. J. Quant. Criminol. **34** (2018)

18. Silva, T.H., et al.: Urban computing leveraging location-based social network data: a survey. ACM Comput. Surv. **52**(1), 17:1–17:39 (2019)
19. Singru, N., Lumain, R., Roldan, C.: Reducing Carbon Emissions from Transport Projects - ADB Strategy 2020 (2010)
20. Siow, E., Tiropanis, T., Hall, W.: Analytics for the internet of things: a survey. ACM Comput. Surv. **51**(4), 74:1–74:36 (2018)
21. Subramanian, R., He, Q., Pascual, M.: Quantifying asymptomatic infection and transmission of COVID-19 in New York city using observed cases, serology, and testing capacity. In: Proceedings of the National Academy of Sciences of the United States of America, vol. 118 (2021)
22. Tao, Y., Hu, X., Choi, D.W., Chung, C.W.: Approximate MaxRS in spatial databases. In: 39th International Conference on Very Large Data Bases 2013, VLDB 2013. vol. 6, pp. 1546–1557 (2013)
23. Wan, L., Ng, W.K., Dang, X.H., Yu, P.S., Zhang, K.: Density-based clustering of data streams at multiple resolutions. ACM Trans. Knowl. Discov. Data (TKDD) **3**(3), 1–28 (2009)
24. Yang, D., Zhang, D., Zheng, V.W., Yu, Z.: Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNS. IEEE Trans. Syst. Man Cybern. Syst. **45**(1), 129–142 (2015)
25. Zhang, J., Zhu, M., Papadias, D., Tao, Y., Lee, D.L.: Location-based spatial queries. In: Proceedings of the 2003 ACM SIGMOD, pp. 443–454. ACM (2003)
26. Zhou, X., Wang, W., Xu, J.: General purpose index-based method for efficient MaxRS query. In: Hartmann, S., Ma, H. (eds.) DEXA 2016. LNCS, vol. 9827, pp. 20–36. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44403-1_2