SURGE: Continuous Detection of Bursty Regions Over a Stream of Spatial Objects

Kaiyu Feng[1,2], Tao Guo[2], Gao Cong[2], Sourav S. Bhowmick[2], Shuai Ma[3]
[1] LILY, Interdisciplinary Graduate School. Nanyang Technological University, Singapore
[2] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[3] SKLSDE Lab, Beihang University, China
{kfeng002@e., tguo001@e., gaocong@, assourav@}ntu.edu.sg, mashuai@buaa.edu.cn

***Abstract*—With the proliferation of location-based services, the generation of massive geo-tagged data opens up new opportunities to address real-world problems. In this paper, we present a novel *continuous bursty region detection* (SURGE) problem that aims to continuously detect a *bursty region* of a given size in a specified geographical area from a stream of spatial objects. The SURGE problem is useful in addressing several real-world challenges such as disease outbreak detection. We propose an exact solution to address the problem, and show the efficiency and effectiveness by conducting experiments on real-world datasets.**

I. Introduction

People often share geo-tagged messages through many social services like *Twitter* and *Facebook*. Each geo-tagged data is associated with a timestamp, a geo-location, and a set of attributes (e.g., tweet content). In this paper, we refer to them as *spatial objects*. With the proliferation of GPS-enabled mobile devices and location-based services, the amount of such spatial objects (e.g., geo-tagged tweets and trip requests using *Uber*) is growing at an explosive rate. Their real-time nature coupled with multi-faceted information and rapid arrival rate in a streaming manner open up new opportunities to address real-world problems.

For example, to tackle the challenge of a variety of virus epidemics like Zika, the U.S. government continuously monitored different areas for possible Zika outbreak. Since early detection of such outbreak is paramount, how can we identify the potential Zika-affected region in real time? As another example, people can use Uber and Lyft to get a ride nowadays. Sometimes people have to wait a long time for a car when the number of car requests significantly surpasses the supply of nearby drivers. Thus, it is beneficial to both passengers and drivers if we can notify idle drivers in real time whenever there is a sudden burst in demand in areas of interest to them.

There are two common themes in the two examples. First, we need to continuously monitor a large volume of spatial objects (e.g., trip requests and disease occurrences) to detect in real time one region that shows relatively large spike in the number of spatial objects (i.e., bursty region) in a given time window. Second, a user needs to specify as input the size $a \times b$ of rectangular-shaped bursty region that one wishes to detect.

In this paper, we refer to the problem embodied in the aforementioned motivating examples as *continuous bursty region detection* (SURGE) problem. Specifically, given a region size $a \times b$ and an interested area A, the aim of the SURGE problem is to continuously detect a region of the specified size in A that demonstrates the *maximum burstiness* from a stream of spatial objects.

In this paper, we propose an exact solution called *cell-*CSPOT to monitor the bursty region. Specifically, we first reduce the SURGE problem to the ***c**ontinuous **b**ursty **p**oint detection (*CSPOT*)* problem. To address the CSPOT problem, we propose a cell-based algorithm with a complexity of $O(|c_{max}|^2 + \log n)$, where $|c_{max}|$ is the maximum number of objects that we search inside a cell, and n is the number of objects. We also conduct extensive experiments to illustrate that our algorithm outperforms the baseline algorithms in an order of magnitude. All the proofs can be found in the full version of this paper [2].

II. Related Work

Burst detection. Our SURGE problem is closely related to efforts on exploring spatial-temporal bursts [5], [4], [7] albeit from different aspects. However, the solution developed in [5] is designed for data warehouse, and it cannot be deployed or adapted to solve the SURGE problem. Lappas et al. [4] takes as input a set of text streams with fixed locations. The proposed solution can only handle a small number of text streams(tens to hundreds) due to its high computational complexity. Zhang et al. [7] define a burst as a cluster of geo-tagged tweets. Moreover, their solution is built over geo-textual stream, while our SURGE is applicable to any kind of spatial stream.

Region search. Our SURGE problem is also closely related to the recent efforts on continuous *MaxRS* problem [1], [3]. Amagata et al. [1] propose the problem of monitoring the *MaxRS* region over spatial data streams. The difference of the SURGE problem from the continuous *MaxRS* problem is that the burst score of the SURGE problem is defined over two consecutive sliding windows, and spatial objects in different windows contribute differently to the burst score. Though their solution cannot be directly applied to solve the SURGE problem, we can adapt their solution with some modifications for the SURGE problem. Hussain et al. [3] investigates the *MaxRS* problem on the trajectories of moving objects, where the position of the objects can be estimated. In our problem, the positions of the new arrived objects are unknown a priori.

III. Problem Statement

A. Terminology

A *spatial object* is a triple $o = \langle w, \rho, t_c \rangle$, where w is the weight, ρ is its location, and t_c is the creation time. For example, a geo-tagged tweet is a spatial object. The *weight* could be the text relevance to a set of query keywords.

We next introduce two consecutive time-based sliding windows, namely *current* and *past windows*. Given a window size $|W|$, the *current window* W_c and the *past window* W_p are two consecutive time periods of length $|W|$ from present time t, i.e., $W_p = (t - 2|W|, t - |W|]$, $W_c = (t - |W|, t]$.

Given a region r and a sliding window W, we define its score as the summation of the objects in $O(r, W)$, normalized by W's length, i.e., $f(r, W) = \frac{\sum_{o \in O(r,W)} o.w}{|W|}$, where $O(r, W) = \{o | o.\rho \in r \wedge o.t_c \in W\}$ is the set of spatial objects which are created in W and located in r.

B. Problem Statement

We first define the *burst score*. Intuitively, the *burst score* of a region r reflects the variation in the spatial objects in r in recent period. This motivates us to design the burst score based on the current and past windows.

We consider the following two factors in our burst score: (a) The score of the region w.r.t. the current window, i.e. $f(r, W_c)$, which measures the *significance*, and (b) the increase in the score of the region between the current window and the past window, i.e., $\max(f(r, W_c) - f(r, W_p), 0)$, which measures the *burstiness*.

We now formally define the burst score as follows.

Definition 1: **Burst Score.** Given a region r, we define its burst score $\mathcal{S}(r)$ as: $\mathcal{S}(r) = \alpha \max(f(r, W_c) - f(r, W_p), 0) + (1-\alpha) f(r, W_c)$, where $\alpha \in [0, 1)$ is a parameter that balances the significance and the burstiness.

We are now ready to formally define the SURGE problem.

Definition 2: **Continuous Bursty Region Detection (SURGE) Problem.** Given a stream of spatial objects $\mathcal{O}$, a preferred area A, a query rectangle of size $a \times b$, and the length $|W|$ of the current and past windows, the **SURGE problem** aims to continuously detect the position of the region r of size $a \times b$ in A with the maximum burst score. The region r is referred to as the **bursty region**.

IV. Solution

The SURGE problem is challenging to address due to the following reasons. First, given a snapshot of the stream, the bursty region can be located at any point, which is infinite. Second, we need to design effective pruning techniques to avoid frequent recomputation of the bursty region given the high arrival rate of the spatial stream.

In this section, we present a solution to address the SURGE problem. We first introduce the ***c**ontinuous **b**ursty **p**o**i**nt de**t**ection* (CSPOT) problem and show that we can reduce the SURGE problem to the CSPOT problem (Section IV-A). Then we present an algorithm to address the CSPOT problem efficiently (Section IV-B and IV-C).

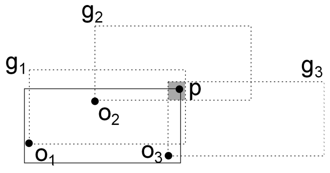


Fig. 1: Reduce to cSPOT problem

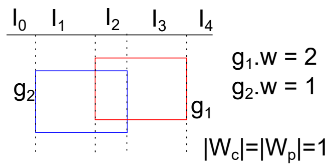


Fig. 2: Illustration of bursty point detection.

A. The cSPOT Problem

Definition 3: **Rectangle Object.** A rectangle object $g = \langle w, \rho, t_c \rangle$ is a rectangle of size $a \times b$, where $g.w$ is its weight, $g.\rho$ is the location of its left-bottom corner, and $g.t_c$ is the creation time of g.

With a slight abuse of notation, we continue to use $f(p, W)$ and $\mathcal{S}(p)$ to denote the score of a point p w.r.t. the window W, and the burst score of p, respectively. Let $G(p, W) = \{g | g.t_c \in W \wedge p \in g\}$. We define $f(p, W) = \frac{\sum_{g \in G(p,W)} g.w}{|W|}$. We define the burst score of a point by following the definition of burst score of a region in Section III.

We are now ready to formally define the CSPOT problem.

Definition 4: **cSPOT Problem.** Consider a stream of rectangle objects $\mathcal{G}$, a parameter α, as well as the current window W_c and past window W_p. The *Continuous Bursty Point Detection* (CSPOT) problem aims to keep track of a point p in the space, such that its burst score $\mathcal{S}(p)$ is maximized. A point p with the maximum score is referred to as **bursty point**.

We illustrate the reduction from the SURGE problem to the CSPOT problem with the example in Figure 1. Assume that $o_1, \ldots, o_3$ are all in A. For each spatial object $o_i, i \in [1, 3]$, we generate a rectangle object g_i of size $a \times b$ with o_i as the left-bottom corner such that $o.t_c = g.t_c$ and $g.\rho = o.\rho$. We next show the relationship between the SURGE problem and the reduced CSPOT problem.

Theorem 1: Given a snapshot of the stream, the bursty point p in the reduced CSPOT problem is the top-right corner of the bursty region in the original SURGE problem.

According to Theorem 1, we can address the SURGE problem by solving the reduced CSPOT problem. Note that in the CSPOT problem, the edges of the rectangles divide the space into $O(n^2)$ disjoint regions[6]. This justifies the reduction: We only need to consider the $O(n^2)$ disjoint regions in the CSPOT problem instead of the infinite points in the SURGE problem.

The reduction is inspired by [6]. However, the techniques designed in [6] cannot be utilized to search for the bursty point at a snapshot in the CSPOT problem.

B. Detecting Bursty Point on a Snapshot

We next present a sweep-line based algorithm called SL-CSPOT to detect the bursty point in the CSPOT problem.

The high level idea of the SL-CSPOT algorithm is as follows. We use a horizontal line, referred to as the sweep-line, to scan the space top-down. The sweep-line is divided into $2n + 1$ intervals at most by the vertical edges of the n rectangle objects. For instance, in Figure 2, the vertical edges of the two rectangles divide the sweep-line into 5 intervals, $\{I_0, \ldots, I_4\}$.

Algorithm 1: SL-CSPOT Algorithm

```
Input: A set of rectangle objects $G$
Output: A bursty point $p$
1 $p = null$;
2 while sweep-line meets an horizontal edge of a rectangle $g$ do
3     $I_i, \dots, I_j \leftarrow$ the intervals covered by $g$;
4     for interval $I \in \{I_i, \dots, I_j\}$ do
5         Update $I.f_c$, $I.f_p$ and $I.\mathcal{S}$;
6         if $I.\mathcal{S} > \mathcal{S}(p)$ then
7             $p \leftarrow$ a point beneath $I$, and between the
              sweep-line and next horizontal edge;
8 return $p$;
```

Algorithm 2: *Cell*-CSPOT Algorithm

```
Input: An event $e = \langle g, l \rangle$
Output: A bursty point
1 $C_g \leftarrow$ cells that are overlapped with $g$;
2 for $c \in C_g$ do
3     Update $U(c)$ using Eqn 1, 2, and status of $c.p$ using
      Lemma 3;
4 $c \leftarrow \arg\max U(c)$;
5 while $c.p$ is invalid do
6     $c.p \leftarrow$ SL-CSPOT($c$);
7     $U_d(c) = \mathcal{S}(c.p)$;
8     $c \leftarrow \arg\max U(c)$;
9 return $c.p$
```

For each interval I, we use $I.f_c$ and $I.f_p$ to denote the score for the points on the interval I w.r.t. the current and past windows, respectively. We use $I.\mathcal{S}$ to denote the burst score of such points. For any interval I_i, the set of rectangles which can cover interval I_i changes when the sweep line meets the top or bottom edge of a rectangle which can cover I_i, and its burst score $I_i.\mathcal{S}$ is updated accordingly. Consider the example in Figure 2. When the sweep-line first meets the top edge of g_1, intervals I_2 and I_3 will be covered by g_1. Assume g_1 is in W_c. Then $I_2.f_c$ and $I_3.f_c$ will be increased by $\frac{g_1.w}{|W_c|} = 2$. Consequently, the burst scores of I_2 and I_3 are updated as $I_i.S = 2$ for $i \in [2, 3]$. When the sweep-line meets the end of the space, a point with the maximum burst score during the sweeping process is returned as the bursty point. The procedure is outlined in Algorithm 1.

Time Complexity. The complexity of Algorithm 1 is $O(n^2)$, where n is the number of rectangles in the space.

C. Handling the Stream

We next present how to continuously detect the bursty point. Note that there are three kinds of events that may change the bursty point: (1) a **new event**, i.e., a new object enters the current window, (2) a **grown event**, i.e., an existing object moves from the current window to the past window, and (3) an **expired event**, i.e., an existing object leaves the past window. We use a tuple $e = \langle g, l \rangle$ to denote an event, where g is the rectangle object, and l is one status from $\{New, Grown, Expired\}$ to indicate the type of the event.

When an event happens, it only affects the burst score of the points inside the rectangle of the event. Motivated by this locality property, we propose a cell-based algorithm called the *Cell*-CSPOT algorithm.

We divide the space into cells of equal size $a \times b$. The high level idea of our cell-based lazy update strategy is as follows: For each cell, we maintain a burst score upper bound for the points inside the cell (to be discussed in Section IV-C1). Whenever an event happens, we first update the upper bound of the overlapped cells. A cell is searched only if its upper bound is higher than the current maximum burst score.

In addition, to reuse the result of Algorithm 1 from previous computations, we record the point returned by Algorithm 1 for each cell which is called *candidate point*. The status of each candidate point is either *valid* or *invalid*. If the candidate point of a cell is guaranteed to have the maximum burst score in the cell, its status is *valid*. Otherwise, the status is set to invalid. We do not need to invoke Algorithm 1 to search a cell if its candidate point is valid (Section IV-C2).

Algorithm 2 outlines the procedure. It takes as input an event $e = \langle g, l \rangle$, and reports a bursty point in the space. The algorithm first locates the set C_g of cells that overlap with g (line 1). Then for each cell c in C_g, it updates its upper bound based on Equations 1, and 2 (Section IV-C1), and determines the status of the candidate point *c.p* based on Lemma 3 (Section IV-C2) (line 3). Then it accesses the cells in descending order of their upper bounds $U(c)$ iteratively (lines 4–8). In each iteration, if the candidate point *c.p* is *invalid*, we search the cell and update *c.p* (line 6) and the upper bound (line 7). Otherwise *c.p* is *valid*, and this indicates that *c.p* has the maximum burst score and we can terminate the process and report *c.p* as the result.

Complexity Analysis. The time complexity of Algorithm 2 is $O(|c_{max}|^2 + \log n)$, where $|c_{max}|$ is the maximum number of rectangle objects in a cell, and n is the number of rectangle objects in W_c and W_p. The space cost of Algorithm 2 is $O(n)$.

1) Upper Bound Estimation: Next, we present the details about estimating the upper bound for a cell.

Static Upper Bound. According to the Definition 1, only the rectangle objects in W_c have a positive impact on the burst score. Hence, we can estimate an upper bound burst score for a cell by only utilizing the objects in the current window. We refer to this upper bound as *static upper bound*.

Definition 5: **Static Upper Bound.** For a cell c, its static upper bound is computed as follows:

$$U_s(c) = \sum_{g \in c.G \wedge g.t_c \in W_c} \frac{g.w}{|W_c|} \tag{1}$$

where $c.G$ is a set of rectangle objects overlapped with c.

Lemma 1: For any point p in cell c, we have $\mathcal{S}(p) \le U_s(c)$.

Dynamic Upper Bound. Next, we introduce another way to estimate the upper bound. Specifically, when an event happens, we dynamically update the upper bound. We refer to such upper bound as *dynamic upper bound*.

Whenever we search a cell c on a snapshot i, we found a point p_m with the maximum burst score. The score $\mathcal{S}(p_m)$ is an upper bound for cell c on snapshot i, i.e., $U_d^i(c) = \mathcal{S}(p_m)$.

TABLE I: Datasets.

Datasets	UK	US	Taxi
# of Spatial Objects	1,000,000	1,000,000	1,000,000
Arrival Rate(per hour)	5,747	16,802	18,145

Let $U_d^i(c)$ be the upper bound of cell c on snapshot i when event e_i arrives. When the $(i+1)$-th event $e_{i+1} = \langle g, l \rangle$ happens, we have

$$U_d^{i+1}(c) = \begin{cases} U_d^i(c) + \frac{g.w}{|W_c|} & l \text{ is New,} \\ U_d^i(c) & l \text{ is Grown,} \\ U_d^i(c) + \alpha \frac{g.w}{|W_p|} & l \text{ is Expired} \end{cases} \quad (2)$$

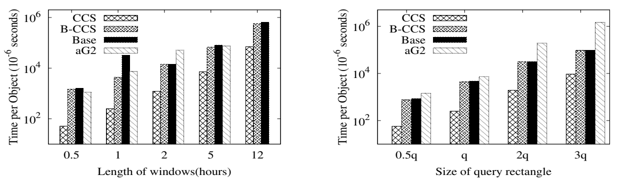
Lemma 2: Consider a cell c. For any point p in c, we have $\mathcal{S}(p) \leq U_d(c)$ after e happens.

We now combine the two bounds for a tighter upper bound.

Definition 6: **Upper bound for cell.** For a cell c, we define its upper bound $U(c)$ as $U(c) = \min(U_s(c), U_d(c))$.

2) Candidate Point Maintenance: In order to reuse the result of invoking Algorithm 1, for each cell c, we maintain a candidate point, denoted by $c.p$, to record the point returned by Algorithm 1. The candidate point has two possible status as introduced before. We next present Lemma 3, which is employed to determine the status of a candidate point.

Lemma 3: Let $c.p$ be a point with the maximum burst score in cell c currently. Consider an event $e = \langle g, l \rangle$. After e happens, if either (1) e is either new or expired, g can cover $c.p$, and $f(c.p, W_c) - f(c.p, W_p) > 0$, or (2) e is grown object and g cannot cover $c.p$, then the point $c.p$ still has the maximum burst score.



(a) Varying $|W|$. (b) Varying $a \times b$.

Fig. 3: Runtime of CCS, B-CCS, Base and aG2 on US.

V. Experimental Study

A. Experimental Setup

Datasets. We conduct experiments on three public real-life datasets. The details of the datasets are reported in Table I. For each dataset, the weight of each spatial object is randomly chosen from [1, 100] with a uniform distribution.

Algorithms. We evaluate the following algorithms. (a) *Cell*-cSpot, denoted by CCS; (b) An approach that only utilizes the static upper bound, denoted by B-CCS; (c) An approach that does not use any upper bound estimation technique, denoted by Base; and (d) a modified version of aG2 [1].

The experiments are run on a machine with a 2.70GHz CPU and 64GB of memory running Ubuntu. The algorithms are implemented in C++ complied with GCC 4.8.2.

TABLE II: Ratio of rectangle messages that trigger a search.

Taxi	Window (mins)	1	5	10	20	30
	CCS	4.85%	3.20%	2.56%	2.13%	1.95%
	B-CCS	92.63%	78.30%	70.00%	62.07%	57.90%
UK	Window (hours)	0.5	1	2	5	12
	CCS	0.34%	0.27%	0.23%	0.37%	0.48%
	B-CCS	37.79%	28.23%	22.76%	21.64%	14.57%
US	Window (hours)	0.5	1	2	5	12
	CCS	0.60%	0.68%	0.70%	0.52%	0.60%
	B-CCS	64.21%	52.29%	35.13%	9.0%	20.90%

B. Experimental Results

Runtime Performance. We first compare the proposed algorithm against the three baseline algorithms. Figure 3 (a) and (b) report the average runtime of the four methods for processing one spatial object as we vary the size of sliding windows and the size of the query rectangle, respectively. We observe that CCS outperforms aG2 by an order of magnitude. We also find that aG2 run out of the 64 GB memory on US when the current window and past window are both set as 12 hours, as there are too many spatial objects in the two sliding windows.

Usefulness of Upper Bound. Next, we evaluate the usefulness of the method for upper bound estimation in CCS. In this set of experiments, we process 1,000,000 new objects and report how many rectangles trigger a search. The results are reported in Table II. Clearly, only a small portion of rectangle messages (2%-5% for Taxi, and less than 1% for US and UK) trigger a search in CCS compared with B-CCS. This is because CCS can estimate a much tighter upper bound for cells. Thus, many cells are eliminated from further checking.

VI. Conclusions

In this paper, we have proposed the SURGE problem aiming at continuously detecting the bursty region in real time. We have proposed a grid-based solution with effective pruning techniques to address the SURGE problem. Experiments on real-life datasets show the efficiency and effectiveness of our proposed solution.

Acknowledgments. This work is supported by Singapore MOE Tier-2 grant (MOE2016-T2-1-137), MOE Tier-1 grant (RG31/17), and a grant from Microsoft. This work is also supported by NSFC under the grant 61772537. Shuai is supported in part by NSFC U1636210 & 61421003.

References

[1] D. Amagata and T. Hara. Monitoring maxrs in spatial data streams. In *EDBT*, pages 317–328, 2016.

[2] K. Feng, T. Guo, G. Cong, S. S. Bhowmicks, and S. Ma. Surge: Continuous detection of bursty regions over a stream of spatial objects. *arXiv preprint arXiv:1709.09287*, 2017.

[3] M. M.-u. Hussain, G. Trajcevski, K. A. Islam, and M. E. Ali. Towards efficient maintenance of continuous maxrs query for trajectories. In *EDBT*, 2017.

[4] T. Lappas, M. R. Vieira, D. Gunopulos, and V. J. Tsotras. On the spatiotemporal burstiness of terms. *PVLDB*, 5(9):836–847, 2012.

[5] M. Mathioudakis, N. Bansal, and N. Koudas. Identifying, attributing and describing spatial bursts. *PVLDB*, 3(1-2):1091–1102, 2010.

[6] 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.

[7] C. Zhang, G. Zhou, Q. Yuan, H. Zhuang, Y. Zheng, L. M. Kaplan, S. Wang, and J. Han. Geoburst: Real-time local event detection in geo-tagged tweet streams. In *SIGIR*, pages 513–522, 2016.