GeoBurst: Real-Time Local Event Detection in Geo-Tagged Tweet Streams

Chao Zhang[1], Guangyu Zhou[1], Quan Yuan[1], Honglei Zhuang[1], Yu Zheng[2,3], Lance Kaplan[4], Shaowen Wang[1], and Jiawei Han[1]
[1]Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[2]Microsoft Research, Beijing, China
[3]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
[4]U.S. Army Research Laboratory, Adelphi, MD, USA
[1]{czhang82, gzhou6, qyuan, hzhuang3, shaowen, hanj}@illinois.edu
[2]yuzheng@microsoft.com [4]lance.m.kaplan.civ@mail.mil

ABSTRACT

The real-time discovery of local events (*e.g.*, protests, crimes, disasters) is of great importance to various applications, such as crime monitoring, disaster alarming, and activity recommendation. While this task was nearly impossible years ago due to the lack of timely and reliable data sources, the recent explosive growth in geo-tagged tweet data brings new opportunities to it. That said, how to extract quality local events from geo-tagged tweet streams *in real time* remains largely unsolved so far.

We propose GEOBURST, a method that enables effective and real-time local event detection from geo-tagged tweet streams. With a novel authority measure that captures the geo-topic correlations among tweets, GEOBURST first identifies several *pivots* in the query window. Such pivots serve as representative tweets for potential local events and naturally attract similar tweets to form candidate events. To select truly interesting local events from the candidate list, GEOBURST further summarizes continuous tweet streams and compares the candidates against historical activities to obtain spatiotemporally bursty ones. Finally, GEOBURST also features an updating module that finds new pivots with little time cost when the query window shifts. As such, GEOBURST is capable of monitoring continuous streams in real time. We used crowdsourcing to evaluate GEOBURST on two real-life data sets that contain millions of geo-tagged tweets. The results demonstrate that GEOBURST significantly outperforms state-of-the-art methods in precision, and is orders of magnitude faster.

Keywords

Twitter; tweet; local event; event detection; social media

1. INTRODUCTION

A local event (*e.g.*, protest, crime, disaster, sport game) is an unusual activity bursted in a local area and within specific duration while engaging a considerable number of participants. The real-time detection of local events was nearly impossible years ago due to the lack of timely and reliable data sources, yet the recent explosive growth in geo-tagged tweet data brings new opportunities to it. With the ubiquitous connectivity of wireless networks and the wide proliferation of mobile devices, more than 10 million geo-tagged tweets are created in the Twitterverse every day [1]. Each geo-tagged tweet, which contains a text message, a timestamp, and a geo-location, provides a unified 3W (*what, when*, and *where*) view of the user's activity. Even though the geo-tagged tweets account for only about 2% of the entire tweet stream [16], their sheer size, multi-faceted information, and real-time nature make them an invaluable source for detecting local events. For example, when the tragic 2011 Tohoku Earthquake hit Japan on March 11th 2011, thousands of related geo-tagged tweets were created instantly; and when the Baltimore Riot took place in April 2015, many people posted geo-tagged tweets to broadcast it right on the spot.

We aim to achieve *real-time local event detection* from geo-tagged tweet streams. This task is important as it can underpin various applications. Take disaster alarming as an example. By detecting emergent disasters (*e.g.*, earthquakes, fires) in real time, we can send alarms to the populace at the very first moment when these disasters outbreak. Such alarms can be much faster than traditional news media [9, 24, 30], and thus allow for timely response that avoids huge life and economic losses. As another example, the detected local events can be easily filtered by a few keywords for activity recommendation. Consider a user who is interested in sport games, movies, and music festivals. With proper filtering keywords, the detector can continuously feed the user with such activities in the city. As such, she can easily learn about what is happening around and decide what to do.

Despite its practical importance, real-time local event detection in the geo-tagged tweet stream is nontrivial because it introduces several unique challenges: (1) *Integrating diverse types of data.* The geo-tagged tweet stream involves three different data types: location, time, and text. Considering the totally different representations of those data types and the complicated correlations among them, how to effectively integrate them for local event detection is challenging. (2) *Extracting interpretable events from overwhelming noise.* Existing studies have revealed that about 40% tweets are just pointless babbles [2], and even those event-related tweets are mostly short and noisy. As truly interesting local events are buried in massive irrelevant tweets, it is nontrivial to accurately identify them and describe in an interpretable way. (3) *On-line and real-time detection.* When a local event outbreaks, it is key to report the event instantly to allow for timely actions. As massive geo-tagged

SIGIR '16, July 17-21, 2016, Pisa, Italy

DOI: http://dx.doi.org/10.1145/2911451.2911519

tweets stream in, the detector should work in an on-line and real-time manner instead of a batch-wise and inefficient one.

A number of studies [6, 12, 13, 28, 17, 5] have investigated event detection in Twitter. Although these techniques have demonstrated inspiring results in detecting global events, they are inapplicable to detecting local events. Unlike global events that are bursty in the entire stream, local events are "bursty" in a small geographical region and involve a limited number of tweets. Such local bursts cannot be readily captured by global event detection methods. Recently, a few methods tailored for local event detection [15, 10, 7, 3] have been introduced. The state-of-the-art method EVENTWEET [3] extracts the keywords that are both temporally bursty and spatially localized, and then clusters those keywords into events based on spatial distributions. Unfortunately, the quality of the result events is limited because it does not model the semantic correlations between keywords. Furthermore, EVENTWEET cannot detect local events in real time, because it partitions the stream into fixed-width time windows and the detection is triggered only when the current window is saturated.

We propose GEOBURST, an effective and real-time local event detector. Our key insight is that, a local event usually leads to a considerable number of geo-tagged tweets around the occurring place (*e.g.*, many participants of a protest may post tweets on the spot). As such tweets are geographically close and semantically coherent, we call them a geo-topic cluster and consider it as a potential local event. Nevertheless, not necessarily does every geo-topic cluster correspond to a local event because (1) the activity could be just routine in that region instead of an unusual event, *e.g.*, many shopping-related tweets are posted on the 5th Avenue in New York every day; and (2) the activity may be geographically scattered instead of localized, *e.g.*, a popular TV show may result in several geo-topic clusters in different regions. Hence, we should also carefully measure the spatiotemporal burstiness of each geo-topic cluster to identify true local events.

Motivated by the above, the first step of GEOBURST finds all geo-topic clusters in the query window as candidate events. Specifically, we compute a tweet's geo-topic authority by combining the geographical and semantic contributions from its similar tweets, where the geographical side is measured using a kernel function, and the semantic side is captured using random walk on a keyword co-occurrence graph. We then design an authority ascent process to identify all pivot tweets, which are essentially authority maxima in the geo-topic space. Such pivot tweets naturally attract similar tweets to form geo-topic clusters.

The second step of GEOBURST ranks all the candidates based on spatiotemporal burstiness. For this purpose, we continuously summarize the stream and store the summaries in a space-efficient structure called activity timeline. The stored summaries, which describe the typical activities in different regions, serve as background knowledge to measure the spatiotemporal burstiness of geo-topic clusters and select out local events.

Besides extracting local events from an ad-hoc query window, GEOBURST also features an updating module that enables continuous monitoring of the stream. As new geo-tagged tweets stream in, GEOBURST can shift the query window and update the results in real time. The updating incurs little time cost because authority computation, which is the most time-consuming operation in the framework, can be completed by subtracting the contributions of outdated tweets and emphasizing the contributions of new ones.

To summarize, we make the following contributions:

1. We design GEOBURST for local event detection in the geo-tagged tweet stream. The effectiveness of GEOBURST is underpinned by a novel pivot seeking process that generates candidate events, along with a ranking module that measures spatiotemporal burstiness based on stream summarization.
2. With the additive property of the authority score, we design an updating module for GEOBURST. It fast updates the event list when the query window shifts, and thus enables real-time and continuous local event detection.
3. We perform extensive experiments on millions of geo-tagged tweets in two different cities, and evaluate the results using a crowdsourcing platform. Our results demonstrate that GEOBURST significantly outperforms state-of-the-art methods in precision, and runs orders of magnitude faster.

2. RELATED WORK

Global Event Detection. Global event detection aims at extracting events that are bursty and unusual in the entire tweet stream. Existing approaches to this end can be classified into two categories: *document-based* and *feature-based*.

Document-based approaches consider each document as a basic unit and group similar documents to form events. Allan *et al.* [6] perform single-pass clustering of the stream, and use a similarity threshold to determine whether a new document should form a new topic or be merged into an existing one. Aggarwal *et al.* [5] also detect events by continuously clustering the tweet stream, but their similarity measure considers both tweet content relevance and user proximity. Sankaranarayanan *et al.* [25] train a Naïve Bayes filter to identify news-related tweets, and cluster them based on TF-IDF similarity. They also enrich each piece of news with location information by extracting geo-entities.

Feature-based approaches [12, 13, 21, 28, 17] identify a set of bursty features (*e.g.*, keywords) from the stream and cluster them into events. Fung *et al.* [12] model feature occurrences with binomial distribution to extract bursty features. He *et al.* [13] construct the time series for each feature and perform Fourier Transform to identify bursts. Weng *et al.* [28] use wavelet transform and auto-correlation to measure word energy and extract high-energy words. Li *et al.* [17] segment each tweet into meaningful phrases and extract bursty phrases based on frequency, which are clustered into candidate events and further filtered using Wikipedia.

The above methods are all designed for detecting global events that are bursty in the entire stream. As aforementioned, a local event is usually bursty in a small geographical region instead of the entire stream. Hence, directly applying these methods to the geo-tagged tweet stream would miss many local events.

There has also been work [24, 22, 19] on detecting specific types of events. Sakaki *et al.* [24] investigate real-time earthquake detection. A classifier is trained to judge whether an incoming tweet is related to earthquake or not, and an alarm is released when the number of earthquake-related tweets is large. Li *et al.* [19] detect crime and disaster events (CDE) with a self-adaptive crawler that dynamically retrieves CDE-related tweets. Different from those studies, we aim to detect all kinds of local events from the stream.

Local Event Detection. Foley *et al.* [11] use distant supervision to extract local events from Web pages, but the proposed method can only extract local events that are well advertised in advance on the Web. Watanabe *et al.* [27] and Quezada *et al.* [23] study location-aware events in the social media, but their major focus is on geo-locating tweets/events, whereas we aim to automatically extract local events from raw geo-tagged tweets.

Krumm *et al.* [15] propose the detection of spatiotemporal spikes in the tweet stream as local events. Nevertheless, their approach can only detect events for pre-defined rigid time windows (*e.g.*, 3-6

pm, 6-9 pm), because it discretizes time and compares the number of tweets in the same bin across different days. It supports neither ad-hoc query windows nor real-time detection.

Chen *et al.* [7] extract events from geo-tagged Flickr photos. By converting the spatiotemporal distribution of each tag into a 3-dimensional signal, they perform wavelet transform to extract spatiotemporally bursty tags, and clusters those tags into events based on co-occurrence as well as spatiotemporal distributions. Such a method, however, can only detect local events in batch manner.

The most relevant work to our task is by Abdelhaq *et al.* [3]. They propose EVENTWEET, which detects local event in a time window with four steps: (1) examine several previous windows to extract bursty words; (2) compute the spatial entropy of each bursty word and select localized words; (3) cluster localized words based on spatial distribution; and (4) rank the clusters based on features such as burstiness and spatial coverage. Unfortunately, EVENTWEET suffers from two drawbacks. First, the clustering of localized keywords is merely based on spatial distribution without considering tweet content. It results in irrelevant keywords in the same cluster, and cannot distinguish different events that occur at the same location. Second, although EVENTWEET is an online method, it is incapable of detecting local events in real time, as the detection is triggered only when the current window is saturated.

3. PRELIMINARIES

In this section, we formulate the real-time local event detection problem, and then explore several of its characteristics, which motivate the design of GEOBURST.

Problem description. Let $\mathcal{D} = (d_1, d_2, \ldots, d_n, \ldots)$ be a continuous stream of geo-tagged tweets that arrive in chronological order. Each tweet d is a tuple $\langle t_d, l_d, E_d \rangle$, where t_d is its post time, l_d is its geo-location, and E_d is a bag of keywords. For each tweet, we use an off-the-shelf tool[1] to extract entities and noun phrases as its keywords. Note that such preprocessing does not affect the generality of our method, and one can also represent each tweet message as a bag of uni-grams for simplicity.

Consider a query time window $Q = [t_s, t_e]$ where t_s and t_e are the start and end timestamps satisfying $t_{d_1} \leq t_s < t_e \leq t_{d_n}$. The local event detection problem consists of two sub-tasks: (1) extract from $\mathcal{D}$ all the local events that occur during Q; and (2) monitor the continuous stream $\mathcal{D}$ and update the local event list in real time as Q shifts continuously.

GEOBURST overview. In practice, a local event often results in a considerable number of relevant tweets around its occurring location. Take Figure 1 as an example. Suppose a protest occurs on the 5th Avenue in New York, many participants may post tweets on the spot to express their attitude, with keywords such as "protest" and "rights". We call such a set of tweets a *geo-topic cluster* as they are geographically close and semantically coherent.

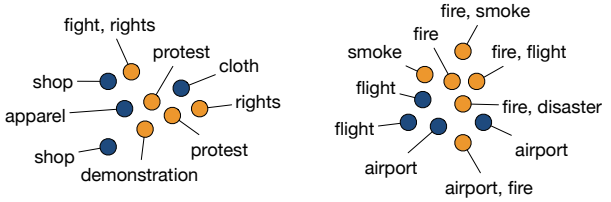


Figure 1: An illustration of geo-topic clusters.

Nevertheless, not necessarily does every geo-topic cluster correspond to a local event even if the cluster has a large size. First, the activity can be just routine in that region. Continue with the example in Figure 1. On almost every day, we can observe many shopping-related tweets on the 5th Avenue. Although such tweets also form a geo-topic cluster, they do not reflect any unusual activities. Second, the cluster may correspond to a global event instead of a local one. For instance, when a popular TV show like "Game of Thrones" goes online, we can observe geo-topic clusters discussing about it in different regions. Such geo-topic clusters do not correspond to local events as well. We thus define *a local event as a geo-topic cluster that shows clear spatiotemporal burstiness*.

Based on the above observations, we first detect all geo-topic clusters in the query window and regard them as candidates — this step ensures high coverage of the underlying local events. The discovery of geo-topic clusters, however, poses several challenges: how to combine the geographical and semantic similarities in a reasonable way? how to capture the correlations between different keywords? and how to generate quality clusters without knowing the suitable number of clusters in advance? To address these challenges, we perform a novel pivot seeking process to identify the centers of geo-topic clusters. Our key insight is that: the spot where the event occurs acts as a pivot that produces relevant tweets around it; the closer we are to the pivot, the more likely we observe relevant tweets. Therefore, we define a geo-topic authority score for each tweet, where the geographical influence among tweets is captured by a kernel function, and the semantic influence by a random walk on a keyword co-occurrence graph. With this authority measure, we develop an authority ascent procedure to retrieve authority maxima as pivots; and each pivot naturally attracts similar tweets to form a quality geo-topic cluster. After discovering the candidate events, we rank them by spatiotemporal burstiness. To this end, we continuously cluster the stream to obtain summaries of regional activities at different timestamps, and store the summaries in a space-efficient structure called activity timeline. The stored summaries serve as background knowledge that enables us to measure the spatiotemporal burstiness of any candidate.

GEOBURST also includes a module that updates the result list in real time as the query window shifts. It will be shown shortly that, the authority score satisfies an additive property. Hence, instead of finding new pivots from scratch when the query window shifts, we can identify them by simply updating the authority scores and then performing fast authority ascent.

We summarize the framework of GEOBURST in Figure 2. As shown, GEOBURST offers two detection modes. The first is the batch mode, which uses the candidate generator and the ranker to detect local events in a fixed query window. The second is the online mode, at the core of which is an updater that updates pivots in real time as the query window shifts. Meanwhile, the ranker is underpinned by the activity timeline structure, which continuously summarizes the stream to obtain background knowledge.

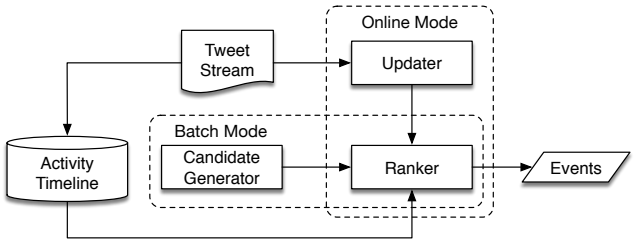


Figure 2: The framework of GEOBURST.

4. THE BATCH MODE

In this section, we describe the batch mode of GEOBURST. As

[1]https://github.com/aritter/twitter_nlp

aforementioned, given a query window Q, the batch mode detects local events in Q by first generating geo-topic clusters as candidate events, and then ranking the candidates by spatiotemporal burstiness. In the sequel, we present the details of these two steps in Sections 4.1 and 4.2, respectively.

4.1 Candidate Event Generation

Let D_Q be the set of tweets falling in Q. The task of candidate generation is to divide D_Q into several geo-topic clusters, such that the tweets in each cluster are geographically close and semantically coherent. As motivated in Section 3, our idea is to view the occurring spot of an event as a pivot and assign each tweet to its corresponding pivot. Below, we introduce a novel geo-topic authority score to define *pivot*, and then develop an authority ascent procedure for pivot seeking.

4.1.1 Pivot

Geographical impact. Given two tweets d and d', we measure the geographical impact of d' to d as

$$G(d' \to d) = K(\|l_d - l_{d'}\|/h),$$

where $K(\cdot)$ is a kernel function, $\|l_d - l_{d'}\|$ is the geographical distance between d and d', and h is the kernel bandwidth. While various kernel functions can be used, we choose the Epanechnikov kernel here due to its simplicity and optimality in terms of bias-variance tradeoff [8]. With the Epanechnikov kernel, $G(d' \to d)$ becomes

$$G(d' \to d) = \begin{cases} c(1 - \|l_d - l_{d'}\|^2/h^2) & \text{if } \|l_d - l_{d'}\| < h \\ 0 & \text{otherwise,} \end{cases}$$

where c is a scaling constant of the Epanechnikov kernel.

Semantic impact. As each tweet message is represented by a bag of keywords, a very straightforward idea for measuring semantic impact is to compute the vector similarity between two tweet messages. Nevertheless, the effectiveness of vector similarity is limited, not only because tweets are short in nature, but also that the dimensions (keywords) are correlated instead of independent. To overcome these drawbacks, we propose a random-walk-based approach to capture semantic impact more effectively.

DEFINITION 1 (KEYWORD CO-OCCURRENCE GRAPH). *The keyword co-occurrence graph for D_Q is an undirected graph $G = (V, E)$ where: (1) V is the set of all keywords in D_Q; and (2) E is the set of edges between keywords, and the weight of an edge (e_i, e_j) is the number of tweets in which e_i and e_j co-occur.*

The keyword co-occurrence graph can be easily built from D_Q. With such a graph, we employ *random walk with restart (RWR)* to define keyword similarity, as it uses the holistic graph structure to capture node correlations. Consider a surfer who starts RWR from the keyword $x_0 = u$. Suppose the surfer is at keyword $x_t = i$ at step t, she returns to u with probability α $(0 < \alpha < 1)$ and continues surfing with probability $1 - \alpha$. If continuing, she randomly moves to i's neighbor j with probability $\mathbf{P}_{ij}$, where $\mathbf{P}$ is the transition matrix of the graph. The stationary distribution of such a process defines the RWR scores from u to all the keywords in V, and the score from u to keyword v, denoted as $r_{u \to v}$, is the probability that the surfer resides on v.

Now, given two tweets d and d', we start RWR from the keywords of d', and define the semantic impact of d' to d as the average probability that the random walk resides on d [20, 29]. Formally, let $E_d = \{e_1, e_2, \ldots, e_m\}$ be the keyword set of d, and $E_{d'} = \{e'_1, e'_2, \ldots, e'_n\}$ the keyword set of d', then the semantic impact from d' to d is

$$S(d' \to d) = \frac{1}{mn} \sum_{e \in E_d} \sum_{e' \in E_{d'}} r_{e' \to e}. \quad (1)$$

Geo-topic authority. Based on geographical and semantic impacts, we measure the *geo-topic authority* of a tweet as follows.

DEFINITION 2 (NEIGHBOR). *Given a tweet d, we say d' is a neighbor of d if d' satisfies $G(d' \to d) > 0$ and $S(d' \to d) > \delta$, where $0 < \delta < 1$ is a pre-specified threshold.*

DEFINITION 3 (AUTHORITY). *Given a tweet $d \in D_Q$, let $N(d)$ be the set of d's neighbors in D_Q. The authority of d is*

$$A(d) = \sum_{d' \in N(d)} G(d' \to d) \cdot S(d' \to d). \quad (2)$$

Given a tweet d, d' is a neighbor of d if it resembles d both geographically and semantically. The set of all neighbors in D_Q form d's neighborhood and contribute to d's authority. We could interpret Equation 2 as follows: an amount of $G(d' \to d)$ energy is distributed from d' to d through random walk on the graph, $G(d' \to d) \cdot S(d' \to d)$ is the amount that successfully reaches d; and d's authority is the total amount of energy that d receives from its neighbors.

Pivot. With Definition 3, we define a *pivot* as an authority maximum.

DEFINITION 4 (PIVOT). *Given a tweet $d \in D_Q$ and its neighborhood $N(d)$, d is a pivot if $A(d) = \max_{d' \in N(d)} A(d')$.*

Consider a local event that occurs at location l. If d is a tweet discussing about that event at l, then d is likely to be surrounded by relevant tweets to become the pivot for that event. The notion of neighborhood plays an important role in Definition 4: it ensures the supporting tweets are both geographically close and semantically relevant. This property leads to different pivots that can distinguish different-semantics events happening at the same location, as well as same-semantics events happening at different locations.

4.1.2 Authority ascent for pivot seeking

Now our task is to find all pivots in D_Q and assign each tweet to its corresponding pivot. We develop an authority ascent procedure for this purpose. As shown in Figure 3, starting from a tweet d_1 as the initial center, we perform step-by-step center shifting. Assuming the center at step t is tweet d_t, we find d_t's neighborhood $N(d_t)$, and the *local pivot* $l(d_t)$ — the tweet having the largest authority in $N(d_t)$. Then we regard $l(d_t)$ as our new center, *i.e.*, $d_{t+1} = l(d_t)$. As we continue such an authority ascent process, the center is guaranteed to converge to an authority maximum. It is because every shift operation increases the authority of the current center, and the authority is upper bounded (there are only a finite number of tweets in D_Q).

Candidate Event Generation. Algorithm 1 depicts the process of finding the pivot for every tweet in D_Q. As shown, we first compute the neighborhood for each tweet $d \in D_Q$ (lines 1-2). Subsequently, we compute the authority of each tweet (lines 3-4), and obtain its local pivot (lines 5-6). So long as the local pivots are obtained, we perform authority ascent to identify the pivot each tweet belongs to. Finally, the tweets having the same pivot are grouped into one geo-topic cluster and returned as a candidate event.

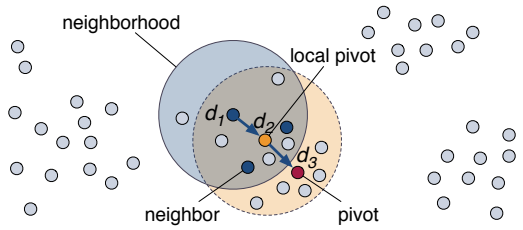


Figure 3: An illustration of the authority ascent process.

Algorithm 1: Pivot seeking.

Input: The tweet set D_Q, the kernel bandwidth h, the semantic threshold δ.
Output: The pivot for each tweet in D_Q.

```
// Neighborhood computation.
```
foreach $d \in D_Q$ **do**
  $N(d) \leftarrow \{d' | d' \in D_Q, G(d' \rightarrow d) > 0, S(d' \rightarrow d) > \delta\}$;
```
// Authority computation.
```
foreach $d \in D_Q$ **do**
  $A(d) \leftarrow d$'s authority score computed from $N(d)$;
```
// Find local pivot for each tweet.
```
foreach $d \in D_Q$ **do**
  $l(d) \leftarrow \arg\max_{d' \in N(d)} A(d')$;
```
// Authority ascent.
```
foreach $d \in D_Q$ **do**
  Perform authority ascent to find the pivot for d;

Fast RWR score computation. In Algorithm 1, while it is easy to compute geographical impact based on tweet location, the challenge is how to compute semantic impact efficiently. A naïve idea is to obtain the RWR score between any two keywords, but such an idea is not efficient as the keyword co-occurrence graph can be large. To address this challenge, we leverage the locality of RWR: given a keyword q, we observe that only a limited number of keywords falling in q's vicinity have large values, while most keywords have extremely small RWR scores. We thus introduce the concept of *keyword vicinity*, which keeps only large enough RWR scores by exploring a small neighborhood around q. Below, we demonstrate how to fast compute the keyword vicinity based on the *Decomposition Theorem* [14].

THEOREM 1. *For a keyword u, let $\mathcal{O}_u$ be the set of u's out-neighbors in G. Given a keyword q, the RWR from u to q satisfies*

$$r_{u \rightarrow q} = \begin{cases} (1-\alpha) \sum_{v \in \mathcal{O}_u} \mathbf{P}_{uv} r_{v \rightarrow q} & \text{if } u \neq q \\ (1-\alpha) \sum_{v \in \mathcal{O}_u} \mathbf{P}_{uv} r_{v \rightarrow q} + \alpha & \text{if } u = q. \end{cases} \quad (3)$$

Theorem 1 says that, the RWR from u to q can be derived by linearly combining the RWR scores of u's out-neighbors, with extra emphasis on q itself. With this theorem, we use a local computation algorithm [20] to obtain q's vicinity. Starting from an initial vicinity, we gradually expand the vicinity and propagate RWR scores among the keywords falling inside. The RWR approximation becomes tighter and tigher as the vicinity expansion continues, and terminates when an error bound ϵ ($0 < \epsilon \ll \delta$) is guaranteed. Algorithm 2 depicts the detailed vicinity computation process. To compute q's vicinity, we maintain two quantities for any keyword u: (1) $s(u)$ is the current RWR score from u to q; and (2) $p(u)$ is the score that needs to be propagated. We use a priority queue to keep $p(u)$ for all the keywords. Every time we pop the keyword

Algorithm 2: Approximate RWR score computation.

Input: The keyword co-occurrence graph G, a keyword q, the restart probability α, an error bound ϵ.
Output: q's vicinity V_q.

$s(q) \leftarrow \alpha, p(q) \leftarrow \alpha, V_q \leftarrow \phi$;
$Q \leftarrow$ a priority queue that keeps $p(u)$ for the keywords in G;
while *Q.peek()* $\geq \alpha\epsilon$ **do**
  $u \leftarrow$ Q.pop();
  for $v \in I(u)$ **do**
    $\Delta s(v) = (1-\alpha) p_{vu} p(u)$;
    $s(v) \leftarrow s(v) + \Delta s(v)$;
    $V_q[v] \leftarrow s(v)$;
    Q.update(v, $p(v) + \Delta s(v)$);
  $p(u) \leftarrow 0$;
return V_q;

u that has the largest to-propagate score, and update the score and to-propagate score for each in-neighbor of u. After that, we set $p(u)$ to zero to avoid redundant propagation. The algorithm terminates when the max element in the priority queue is less than $\alpha\epsilon$, and returns all the keywords that have non-zero RWR scores as q's vicinity. Any keyword u not in q's vicinity must satisfy $r_{u \rightarrow q} < \epsilon$.

THEOREM 2. *Let $\hat{r}_{u \rightarrow q}$ be the approximate RWR score computed by Algorithm 2, then $\hat{r}_{u \rightarrow q}$ satisfies $|r_{u \rightarrow q} - \hat{r}_{u \rightarrow q}| \leq \epsilon$. The time complexity of Algorithm 2 is $O(D_q / \alpha \log 1/(\epsilon\alpha))$, where $D_q = \sum_{u: s_{u \rightarrow q} > \alpha\epsilon} (|I(u)| + \log |V|)$.*

PROOF. See [20] for details. □

4.2 Candidate Ranking

Up to now, we have obtained a set of geo-topic clusters in the query window as candidate events. Nevertheless, as discussed in Section 3, not necessarily does every candidate correspond to a local event because it can be either routine in that region or a global event. In this subsection, we describe GEOBURST's ranking module, at the core of which is the activity timeline structure to facilitate ranking candidates by spatiotemporal burstiness. In what follows, we describe activity timeline construction in Section 4.2.1, and present the ranking function in Section 4.2.2.

4.2.1 Activity timeline construction

The activity timeline aims at summarizing the stream to unveil typical activities in different regions during different time periods. For this purpose, we design a structure called *tweet cluster* (TC), and extend CluStream [4], an effective stream clustering algorithm.

Let S be a set of tweets that are geographically close, its TC maintains the following statistics:

1) $n = |S|$: the number of tweets.
2) $m_l = \sum_{d \in S} l_d$: the sum vector of locations.
3) $m_{l^2} = \sum_{d \in S} l_d \circ l_d$: the squared sum vector of locations.
4) $m_t = \sum_{d \in S} t_d$: the sum of timestamps.
5) $m_{t^2} = \sum_{d \in S} t_d^2$: the squared sum of timestamps.
6) $m_e = \sum_{d \in S} E_d$: the sum dictionary of keywords.

The TC essentially provides a where-when-what summary for S: (1) where: with n, m_l, and m_{l^2}, one can easily compute the mean location and spatial variance for S; (2) when: with n, m_t, and m_{t^2}, one can easily compute the mean time and temporal variance for S; and (3) what: m_e keeps the number of occurrences for each keyword. As we shall see shortly, those fields enable us to estimate the number of keyword occurrences at any location. Moreover, TC satisfies the additive property, *i.e.*, the fields can be easily incremented

if a new tweet is absorbed. Based on this property, we adapt CluStream to continuously clusters the stream into a set of TCs. When a new tweet d arrives, it finds the TC m that is geographically closest to d. If d is within m's boundary (computed from n, m_l, and m_{l^2}, see [4] for details), it absorbs d into m and updates its fields; otherwise it creates a new TC for d. Meanwhile, we employ two strategies [26] to limit the maximum number of TCs: (1) deleting the TCs that are too old and contain few tweets; and (2) merging closest TC pairs until the number of remaining TCs is small enough.

The *activity timeline* is a sequence of clustering snapshots produced by CluStream at different timestamps. As storing the snapshot of every timestamp is unrealistic, we use the pyramid time frame (PTF) structure [4] to achieve both good space efficiency and high coverage of the stream history. The PTF structure imposes finer granularity for recent snapshots and coarser granularity for old ones. It involves two integer parameters: $b > 1$ and $l > 0$. Given a tweet stream $\mathcal{D}$, assume the start timestamp is 0 and the most recent timestamp is T. PTF creates $\lceil \log_b T \rceil$ layers: $0, 1, \ldots, \log_b T$. Each layer i stores the snapshots for the timestamps that can be divided by b^i. If a timestamp matches multiple layers, it is stored in the highest possible layer to avoid redundancy. Further, every layer has a capacity of $b^l + 1$ so that only the latest $b^l + 1$ snapshots are stored. As such, the total number of snapshots in PTF is no larger than $(b^l + 1)\lceil \log_b(T) \rceil$.

EXAMPLE 1. *Suppose the start timestamp of the stream is 0 and the most recent timestamp is 25. Then the maximum layer in the corresponding PTF is $\lfloor \log_2 25 \rfloor = 4$, and the capacity of each layer is 5. The snapshots stored in different layers are:*

Layer 0 : 25 23 21 19 17;
Layer 1 : 22 18 14 10 6;
Layer 2 : 20 12 4;
Layer 3 : 24 8;
Layer 4 : 16.

4.2.2 The ranking function

The snapshots stored in the activity timeline serve as background knowledge for ranking candidate events. To derive the ranking score of candidate C, we quantify the spatiotemporal burstiness of each keyword in C, and then aggregate the burstiness of all the keywords as C's final ranking score.

Temporal burstiness. First, we measure how temporally bursty a keyword $k \in C$ is at the pivot location l_C, by vertically comparing C against the historical activities at l_C. As shown in Figure 4, we retrieve the snapshots in a reference time window R that right precedes the query window Q. Each pair of consecutive snapshots in R corresponds to a *historical activity*, defined as follows.

DEFINITION 5 (HISTORICAL ACTIVITY). *Let s_1 and s_2 be two snapshots at timestamp t_{s_1} and t_{s_2} ($t_{s_1} < t_{s_2}$). The historical activity during the time interval $[t_{s_1}, t_{s_2}]$ is the set of TCs obtained by subtracting s_1 from s_2.*

Let us use an example in Figure 4 to illustrate how we acquire historical activities in the reference window R. As shown, the snapshots s_1, s_2, s_3, s_4 fall in R. For each pair of consecutive snapshots, *i.e.*,$[s_1, s_2]$, $[s_2, s_3]$, $[s_3, s_4]$, we perform snapshot subtraction to obtain the historical activity during the respective time interval. For instance, for the snapshot pair $[s_1, s_2]$, we subtract s_1 from s_2 and obtain the historical activity, represented as a set of TCs: $\{m_1, m_2, m_4, m_6, m_7, m_8\}$. Note that the subtraction of two snapshots can be easily done by matching TC ids and subtracting the fields.

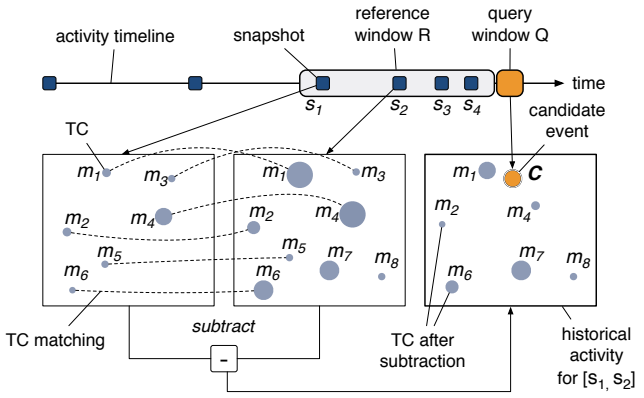


Figure 4: Retrieving historical activities from activity timeline.

Using each historical activity, we can employ kernel density estimation to infer the expected number of occurrences of keyword k at location l_C. Consider the query time window $Q = [t_s, t_e]$ and the historical activity in $[t'_s, t'_e]$. Denote the set of TCs in $[t'_s, t'_e]$ as $\{m_1, m_2, \ldots, m_n\}$. We estimate the number of occurrences of keyword k at location l_C as:

$$N_l(k) = \frac{t_e - t_s}{t_{e'} - t_{s'}} \sum_{i=1}^{n} K(\frac{\|l_C - l_{m_i}\|}{h}) N_{m_i}(k),$$

where $N_{m_i}(k)$ is the number of k's occurrences in m_i, l_{m_i} is the mean location of m_i, and h is the kernel bandwidth. As R contains multiple historical activities, and each can generate an estimation of keyword k's occurrences at location l_C, we obtain a set of estimations, denoted as $\Omega_t = \{\hat{N}_1(k), \hat{N}_2(k), \ldots, \hat{N}_c(k)\}$. Then we use z-score to quantify k's temporal burstiness at l_C:

$$z_t(k) = \frac{N(k) - \mu_{\Omega_t}}{\sigma_{\Omega_t}},$$

where $N(k)$ is k's actual number of occurrences in C, and μ_{Ω_t} and σ_{Ω_t} are the mean and standard deviation of Ω_t.

Spatial burstiness. To measure spatial burstiness, we horizontally compare all the candidates in Q. The rationale is that, among the spatially scattered candidates, a keyword k in candidate C is spatially bursty if k's proportion in C is significantly higher than in other candidates. Given n candidate events $C_1, C_2, \ldots, C_n$, let P_i denote the keyword probability distribution of candidate C_i. With $\Omega_s = \{P_1(k), P_2(k), \ldots, P_n(k)\}$, we compute the spatial burstiness of keyword k in candidate C_i as:

$$z_s(k) = \frac{P_i(k) - \mu_{\Omega_s}}{\sigma_{\Omega_s}},$$

where μ_{Ω_s} and σ_{Ω_s} are the mean and standard deviation of Ω_s.

Ranking function. For each keyword k in a candidate C, we have used vertical comparison against the historical activities to measure its temporal burstiness, and horizontal comparison with the other candidates to measure its spatial burstiness. As the final step, we compute the ranking score of candidate C by aggregating the burstiness of all the keywords in C:

$$s(C) = \sum_{k \in C} w_C(k)(\eta z_t(k) + (1 - \eta) z_s(k)),$$

where η $(0 < \eta < 1)$ is a factor balancing the spatial and temporal burstiness; and $w_C(k)$ is the TF-IDF weight of keyword k.

5. THE ONLINE MODE

In this section, we present the online mode of GEOBURST. Consider a query window Q, let Q' be the new query window after Q shifts. Instead of finding local events in Q' from scratch, the online mode leverages the results in Q and updates the event list with little cost. The updated event list is guaranteed to be the same as directly running batch-mode detection on Q'.

In GEOBURST's two steps, the candidate generation step has a dominating time cost. At the core of the online mode is thus an updating module that finds new pivots in the new window Q' with little overhead. Let D_Q be the tweets falling in Q and D'_Q be the tweets in Q'. We denote by R_Q the tweets removed from D_Q, *i.e.*, $R_Q = D_Q - D'_Q$; and by I_Q the tweets inserted into D_Q, *i.e.*, $I_Q = D'_Q - D_Q$. In the sequel, we design a strategy that finds pivots in D'_Q by just processing R_Q and I_Q.

Recall that, the pivot seeking process first computes the local pivot for each tweet and then performs authority ascent via a path of local pivots. So long as the local pivot information is correctly maintained for each tweet, the authority ascent can be fast completed. The key to avoiding finding pivots from scratch is that, as D_Q is changed to D'_Q, only a number of tweets have their local pivots changed. We call them mutated tweets and identify them by analyzing the influence of R_Q and I_Q.

DEFINITION 6 (MUTATED TWEET). *A tweet $d \in D'_Q$ is a mutated tweet if d's local pivot in D'_Q is different from its local pivot in D_Q.*

For any tweet, it can become a mutated tweet only if at least one of its neighbors has authority change. Therefore, we take a *reverse search* strategy to find mutated tweets. We first identify in D'_Q all the tweets whose authorities have changed. Then for each authority-changed tweet t, we retrieve the tweets that regard t as its neighbor, and update their local pivots. Hence, the key becomes how to find authority-changed tweets. In what follows, we handle R_Q and I_Q to this end.

Handling deletions. The deletion of a tweet $d \in R_Q$ can cause authority change in two ways. First, for the tweets having d as a neighbor in D_Q, their authorities decrease. Second, the keyword co-occurrence graph may evolve because of deleting d. As a result, the vicinities of certain keywords need to be recomputed and the authorities of corresponding tweets may change. The first case can be easily handled due to the additive property of authority. When d is deleted, we simply retrieve the tweets having d as a neighbor in D_Q. For each of those tweets, we subtract d's contribution from the authority score. For the second case, the key is to identify the keywords that need vicinity recomputation. Let us look at an example in Figure 5. If d contains two keywords e_1 and e_2, deleting d would decrease the weight of the edge $[e_1, e_2]$. For any other keywords having e_1 or e_2 in their old vicinities (e_3 and e_4 in this example), we mark them as to-recompute keywords. However, we defer the computation of their vicinities until I_Q is handled to identify the complete set of to-recompute keywords.

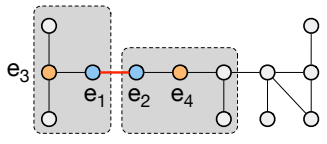


Figure 5: Updating the keyword co-occurrence graph and keyword vicinities.

Handling insertions. A new tweet $d \in I_Q$ can also cause authority changes in two ways: (1) increasing the authority of the tweets that regard d as a neighbor; and (2) making the keyword co-occurrence graph evolve. Here, we need to first deal with the second case to ensure authority computation in the first case is based on the updated keyword vicinities. Similarly, we identify the keywords whose attaching edges have weight change, and mark other keywords that include such keywords in their vicinities. After all the to-recompute keywords are identified, we call Algorithm 2 to obtain their new vicinities. Once the keyword vicinities are updated, we retrieve the affected tweet pairs and update the corresponding authority scores. For the second case, now that the keyword vicinities have already been updated, for the inserted tweet d, we simply find which other tweets having d as their neighbor, and then add d's contribution to their authorities.

6. EXPERIMENTS

We evaluate the empirical performance of GEOBURST in this section. All the algorithms were implemented in JAVA and the experiments were conducted on a computer with Intel Core i7 2.4Ghz CPU and 8GB memory.

6.1 Experimental Setup

Data Set. Our experiments are based on two real-life data sets, both of which are crawled using Twitter Streaming API[2] during 2014.08.01 — 2014.11.30. The first data set, referred to as NY, consists of 9.5 million geo-tagged tweets in New York. After removing the tweets having no entities or noun phrases, we obtain 2.4 million tweets. The second data set, referred to as LA, consists of 9.9 million geo-tagged tweets in Los Angeles, and there are 2.8 million tweets that have entities and/or noun phrases. The reason we choose these two cities is because they have quite different population distributions — the populace of New York is relatively concentrated in Manhattan and Brooklyn, while the populace of Los Angeles is spread out in many different districts.

Compared methods. For comparison, we implemented two existing local event detection methods. The first is EVENTWEET [3], which extracts bursty and localized keywords as features, and then clusters those features based on their spatial distributions. The second is WAVELET [7], which uses wavelet transform to identify spatiotemporally bursty keywords and then clusters them by considering both co-occurrence and spatiotemporal distribution.

Parameters. There are four major parameters in GEOBURST: (1) the kernel bandwidth h; (2) the restart probability α; (3) the RWR similarity threshold δ; and (4) the ranking parameter η for balancing spatial and temporal burstiness. Unless stated explicitly, we set $h = 0.01, \alpha = 0.2, \delta = 0.02$, and $\eta = 0.5$. EVENTWEET partitions the whole space into $N \times N$ small grids. We find N is EVENTWEET's most sensitive parameter, and set $N = 50$ after tuning. For WAVELET, the most sensitive parameters are the granularities for constructing the spatiotemporal signal. After tuning, we set the space partitioning granularity to $\delta_x = 0.1, \delta_y = 0.1$; and the time granularity to $\delta_t = 3$ hours.

6.2 Effectiveness Study

To evaluate effectiveness, we randomly generate 80 query time windows that are non-overlapping during 2014.08.01 — 2014.11.30. Among them, there are 20 3-hour windows, 20 4-hour ones, 20 5-hour ones, and 20 6-hour ones. As all the three methods require a reference window, we use a 5-day reference window right preceding each query. For every query, we run the three methods to retrieve top-5 local events on the two data sets, and upload the results

[2]https://dev.twitter.com/streaming/overview

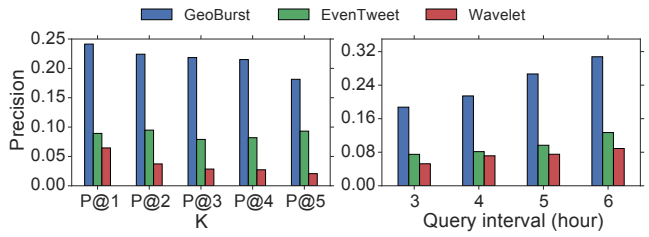
to CrowdFlower[3], a popular crowdsourcing platform, for evaluation. For GEOBURST, we ran both its batch mode and online mode to detect local events in the query window, and found these two modes produce exactly the same results. Thus, we only upload the results produced by the online mode and report its effectiveness.

On CrowdFlower, we represent each event with 5 most representative tweets as well as 10 representative keywords, and ask three CrowdFlower workers to judge whether the event is indeed a local event or not. To ensure the quality of the workers, we label 10 queries for groundtruth judgments on each data set, such that only the workers who can achieve no less than 80% accuracy on the groundtruth can submit their answers. Finally, we use majority voting to aggregate the workers' answers.

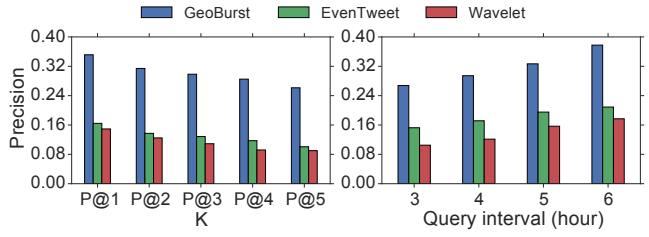
The representative tweets and keywords are selected as follows: (1) For GEOBURST, each event is a cluster of tweets, we select the 5 tweets having the largest authority scores, and the 10 keywords having the largest TF-IDF weights. (2) EVENTWEET represents each event as a group of keywords. We select top-10 keywords in each event. Then we regard the group of keywords as a query to retrieve the top-5 most similar tweets in the query window using the BM25 retrieval model. (3) WAVELET represents an event with both keywords and matching tweets. We simply select the top-5 tweets and the top-10 keywords.

6.2.1 *Quantitative analysis*

After gathering judgments from CrowdFlower, we compute the precision for each method. Figure 6 shows the precisions of the three methods on NY and LA when K and query interval vary. One may notice that the overall precision is not high for all the three methods. This phenomenon is reasonable because the query time windows are generated randomly. There are chances that some query windows fall in a time period (*e.g.*, early morning) during which no local events happened in the city.



(a) Precision comparison (NY).



(b) Precision comparison (LA).

Figure 6: Precision comparison of GEOBURST, EVENTWEET, and WAVELET.

Comparing the three methods, we find that GEOBURST significantly outperforms EVENTWEET and WAVELET on both data sets. The huge improvements indicate the superiority of GEOBURST's two-step scheme: (1) the candidate generation step ensures a good coverage of all potential local events; and (2) the ranking step effectively identifies true local events by carefully measuring spatiotemporal burstiness.

We also observe that the precisions of the three methods all increase with the query interval. It is because a larger query interval is more likely to cover a time period in which certain local events have taken place.

6.2.2 *Illustrative cases*

Now we perform a case study on NY to compare the local events detected by the three methods. We choose the query window to be 7 – 10pm on November 7th 2014 and show the top-3 local events detected by different methods in Figure 7. For each event, we show the messages of the top-3 tweets, highlight the representative keywords, and plot the locations of the member tweets.

Examining the results of GEOBURST, one can see the generated geo-topic clusters are of high quality: the tweets in each cluster are both geographically compact and semantically coherent. Interestingly, GEOBURST can group the tweets that discuss about the topic using different keywords (*e.g.*, "Thai Restaurant" and "Asian Dishes"). This is because the RWR measure effectively captures the subtle semantic correlations between keywords. Another interesting observation is that, the pivot tweet of each cluster is highly interpretable. This is because such high-quality tweets mention most important keywords about the topic and locate closely to the occurring spot, thereby receiving high authority scores.

For the given query, the top two clusters produced by GEOBURST correspond to two local events: (1) the New York Festival of Light, which is held under the Manhattan Bridge; and (2) the basketball game between two NBA teams — Brooklyn Nets and New York Knicks. The third cluster is a group of tweets discussing about dinner instead of any interesting local events. Nevertheless, our investigation into its z-score reveals that there is a huge score gap (16.23 versus 0.73) between the second cluster and the third one. This suggests that one can easily use a z-score threshold to rule out non-event clusters in practice.

EVENTWEET also successfully detects the basketball game between Nets and Knicks. Nevertheless, the interpretability of the result event is not satisfactory, because it just groups bursty keywords based on their spatial distributions. As a result, the keywords in the same cluster may not be semantically coherent. One may notice that there exist geographically faraway tweets in the same cluster. This is because EVENTWEET is a feature-based method that uses only keywords to represent an event. As the matching tweets are retrieved based on keyword similarity, it is possible that some tweets are geographically faraway even if they use the same keywords.

For WAVELET, it also reports the basketball game event, but misses the event of New York Festival of Light. The result clusters are of high quality as WAVELET considers both keyword co-occurrence information and spatiotemporal distribution during the clustering process. Nevertheless, the major drawback of WAVELET is that it tends to miss many local events in the query window. Using wavelet transform for bursty keyword identification, WAVELET is more suitable for extracting influential local events on a tweet collection that spans a long time period, rather than real-time local event detection in an ad-hoc query window.

6.3 Efficiency Study

To study the efficiency of the three methods, we generate 200 random queries with different lengths. For every query, we run each method for 10 times and report the average running time.

6.3.1 *Running time comparison*

In the first set of experiments, we compare the running time of

[3]http://www.crowdflower.com/

	GeoBurst	Is event?
# 1	1. **Festival of Light**! **#nyfol** (@ **The Archway** under the **Manhattan Bridge** in **Brooklyn**, **NY**) 2. **#Lasers and beats** under the **Manhattan Bridge**! **#NewYorkFestivalofLight #NYFOL @ DUMBO** 3. **New York Festival of Lights #nyfol #dumbo @ DUMBO**, **Brooklyn**	**Yes**
# 2	1. **Knicks** vs. **Nets** at **Barclays Center**. @ Barclays Center http://t.co/PILk1xK3Tn 2. **Brooklyn** go hard @ **Barclays Center** http://t.co/iVUsJJ5TNG 3. Let's go **Knicks**! #NETS1107 (@ **Barclays Center** - @brooklynnets for @nyknicks vs @BrooklynNets)	**Yes**
# 3	1. **#Thai Restaurant** #spicythaifood (@ 104 **2nd Avenue** in **New York**, **NY**) 2. The **ASIAN DISHES** here are always my favorite. @ **Ugly Kitchen** 3. **Dinner time** with my family. Suuuuper Nice **Indian RESTAURANT**! @ Malai Marke **Indian Cuisine**.	No
	EvenTweet	**Is event?**
# 1	1. I practiced... Almost time for **Amy Schumer**. **Jennifer** (@ **Carnegie Hall**) https://t.co/HfqfTLmK2y 2. **2014 Gold Glove Awards Ceremony** with **Hall of Famers**, **All-Stars Jay Leno @ The Plaza Hotel** 3. My best attempt at a **selfie** with **Hugh Jackman** after **The River at CITS @ The River on Broadway**	No
# 2	1. **Knicks** vs. **Nets** at **Barclays Center**. @ **Barclays Center** http://t.co/PILk1xK3Tn 2. **Budweiser** brings everyone together #family #nonewfriends @ **Alchemy Tavern**, **Brooklyn** 3. **#Knicks** vs **#nets** with my best gal. @ **Barclays Center Brooklyn** http://t.co/eXXMUKxpIs	**Yes**
# 3	1. **#katespade @ Kate Spade / Jack Spade HQ** http://t.co/g6jiFwyc4M 2. Inspiring **keynote** by **Twitter CEO**, **Dick Costolo** @GirlsWhoCode Gala. http://t.co/yEGh803CuT 3. I wonder if **Jake** from **Statefarm** covers **Jumanji**?	No
	Wavelet	**Is event?**
# 1	1. **Spike**'s face says it all. Sorry **Knicks** fans maybe next time. **@Brooklynnets #Nets @ Barclays Center** 2. **#Knicks** vs **#nets** with my best gal. @ **Barclays Center Brooklyn** http://t.co/eXXMUKxpIs 3. **Brooklyn** go hard @ **Barclays Center** http://t.co/hPooLfa9pQ	**Yes**
# 2	1. **#BEMF** (at **@VerbotenNewyork** in **Brooklyn**, NY) https://t.co/uVgTHNZzSP 2. Starting the **weekend** right. (at **@VerbotenNewyork** in **Brooklyn**, **NY**) https://t.co/cRiPHK1Qdy 3. What a **NIGHT**!! @ **Irving Plaza**	No
# 3	1. **Alex** from target isn't even cute... 2. Thank you **Alex** and **Eric** and **chief Alex** and @nyysteak dad loves his **long bone**. 3. So here I am in **NYC** alone without **Alex**. but I know who i am I'm in the right **direction**.	No

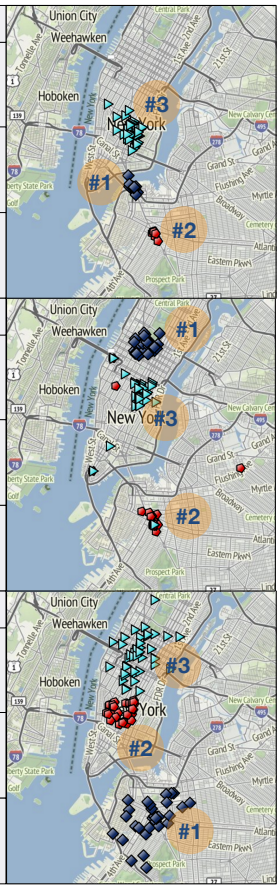
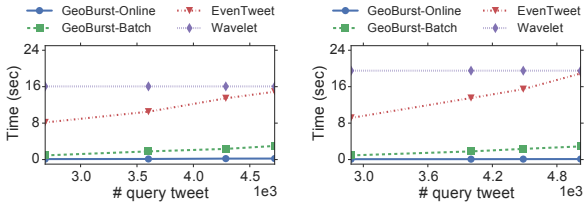


Figure 7: Top-3 local events detected by different methods in New York during 7 – 10pm, November 7th 2014.

GeoBurst against EvenTweet and Wavelet. We run GeoBurst in both batch mode and online mode. Given a query window Q, the batch mode performs candidate generation and ranking in Q; the online mode considers a window Q' that precedes Q by 10 minutes, and finds local events in Q by updating the results in Q'.

Figure 8 shows the running time of the three methods on NY and LA. We observe that GeoBurst is much more efficient than EvenTweet and Wavelet even when in the batch mode. This phenomenon is explained by two facts. First, in the candidate generation step, the approximate RWR computation strategy can effectively speed up the pivot seeking process. Second, in the ranking step, GeoBurst just uses a number of historical activities to compute z-scores, which is very efficient (we found that the running time of the ranking step accounts for less than 1% of GeoBurst's total running time). Meanwhile, the online mode is even much faster than the batch mode. This is expected as the online mode does not need to find pivots from scratch in the time-consuming candidate generation step, but just needs to process the updated tweets and can achieve excellent efficiency.

The major overhead of EvenTweet and Wavelet is due to their space partitioning strategy. Specifically, EvenTweet needs to compute spatial entropy to select localized keywords and perform clustering based on keyword spatial distributions; Wavelet needs to perform wavelet transform on the spatiotemporal signal and compute the spatiotemporal KL-divergence between keywords. One may propose to partition the space at a coarser granularity to improve the running time of the two methods, but that comes with the price of being much less effective. Note that the running time of



(a) Running time (NY). (b) Running time (LA).

Figure 8: Running time v.s. # tweets in the query window.

Wavelet remains almost unchanged when the number of tweets increases. The reason is Wavelet finds events in an augmented window that includes both reference and query tweets. As our queries are at the same scale of several hours, we fixed the length of the reference window to 5 days. Accordingly, the running time of Wavelet is not affected much by the length of the query interval.

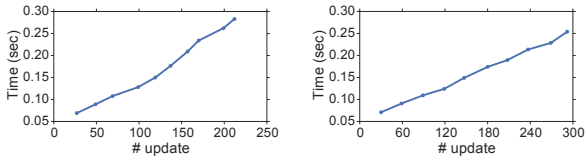
6.3.2 *Throughput Study*

In Figure 9, we report the scalability of GeoBurst's online mode when the number of updates varies:

$$\#\text{updates} = \#\text{deleted tweets} + \#\text{inserted tweets}.$$

To this end, we choose a 3-hour query window Q. Then we use a window Q' that precedes Q by 1, 2, ..., 8, 9, 10 minutes, respectively, and update the results in Q'. One can observe that the running time of the online mode shows good scalability with the number of updates. For example, when there are as many as 212 updates, the online mode takes just 0.282 second to finish on the

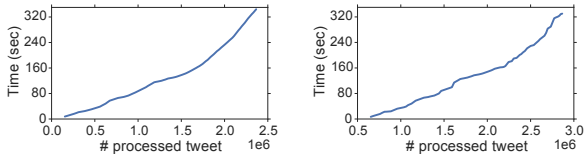
NY data set. Such performance suggests that, GEOBURST's online mode is capable of continuously monitoring the stream and realizing real-time detection.



(a) Time v.s. # update (NY). (b) Time v.s. # update (LA).

Figure 9: Throughput of GEOBURST's online mode.

We proceed to study the throughput of GEOBURST for constructing the activity timeline. We apply GEOBURST to construct activity timeline on NY and LA and periodically record the number of tweets processed so far and the running time. As shown in Figure 10, GEOBURST finished constructing activity timeline for 2.4 million tweets in 341.35 seconds on NY, and 2.8 million tweets in 330.82 seconds on LA, and it scales well with the number of tweets.



(a) Time v.s. # tweet (NY). (b) Time v.s. # tweet (LA).

Figure 10: Throughput of activity timeline construction.

7. CONCLUSION

We studied the problem of real-time local event detection in the geo-tagged tweet stream. We proposed the GEOBURST detector. To the best of our knowledge, GEOBURST is the first method that is capable of extracting highly interpretable local events in real time. GEOBURST first generates candidate events based on a novel pivot seeking process, and then leverages the continuous summarization of the stream as background knowledge to rank the candidates. Our extensive experiments have demonstrated that GEOBURST is highly effective and efficient. The usage of GEOBURST is not limited to Twitter. Rather, any geo-textual social media stream (*e.g.*, Instagram photo tags, Facebook posts) can use GEOBURST to extract interesting local events as well. For future work, it is interesting to extend GEOBURST for handling the tweets that mention geo-entities but do not include exact GPS coordinates. We plan to pursue this direction by augmenting GEOBURST with existing geo-locating techniques [18, 27].

8. ACKNOWLEDGEMENTS

This work was sponsored in part by the U.S. Army Research Lab. under Cooperative Agreement No. W911NF-09-2-0053 (NSCTA), National Science Foundation IIS-1017362, IIS-1320617, and IIS-1354329, HDTRA1-10-1-0120, NSFC (Grant No. 61572488), and Grant 1U54GM114838 awarded by NIGMS through funds provided by the trans-NIH Big Data to Knowledge (BD2K) initiative (www.bd2k.nih.gov), and MIAS, a DHS-IDS Center for Multimodal Information Access and Synthesis at UIUC. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies of the U.S. Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

9. REFERENCES

[1] http://goo.gl/GQF38b.
[2] http://goo.gl/i0Gdol.
[3] H. Abdelhaq, C. Sengstock, and M. Gertz. Eventweet: Online localized event detection from twitter. *PVLDB*, 6(12):1326–1329, 2013.
[4] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, pages 81–92, 2003.
[5] C. C. Aggarwal and K. Subbian. Event detection in social streams. In *SDM*, pages 624–635, 2012.
[6] J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *SIGIR*, pages 37–45, 1998.
[7] L. Chen and A. Roy. Event detection from flickr data through wavelet-based spatial analysis. In *CIKM*, pages 523–532, 2009.
[8] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *ICCV*, pages 1197–1203, 1999.
[9] S. Doan, B.-K. H. Vo, and N. Collier. An analysis of twitter messages in the 2011 tohoku earthquake. In *Electronic Healthcare*, pages 58–66. Springer, 2012.
[10] W. Feng, C. Zhang, W. Zhang, J. Han, J. Wang, C. Aggarwal, and J. Huang. Streamcube: Hierarchical spatio-temporal hashtag clustering for event exploration over the twitter stream. In *ICDE*, pages 1561–1572, 2015.
[11] J. Foley, M. Bendersky, and V. Josifovski. Learning to extract local events from the web. In *SIGIR*, pages 423–432, 2015.
[12] G. P. C. Fung, J. X. Yu, P. S. Yu, and H. Lu. Parameter free bursty events detection in text streams. In *VLDB*, pages 181–192, 2005.
[13] Q. He, K. Chang, and E.-P. Lim. Analyzing feature trajectories for event detection. In *SIGIR*, pages 207–214, 2007.
[14] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.
[15] J. Krumm and E. Horvitz. Eyewitness: Identifying local events via space-time signals in twitter feeds. In *SIGSPATIAL*, 2015.
[16] K. Leetaru, S. Wang, G. Cao, A. Padmanabhan, and E. Shook. Mapping the global twitter heartbeat: The geography of twitter. *First Monday*, 18(5), 2013.
[17] C. Li, A. Sun, and A. Datta. Twevent: segment-based event detection from tweets. In *CIKM*, pages 155–164, 2012.
[18] G. Li, J. Hu, J. Feng, and K.-l. Tan. Effective location identification from microblogs. In *ICDE*, pages 880–891, 2014.
[19] R. Li, K. H. Lei, R. Khadiwala, and K.-C. Chang. Tedas: A twitter-based event detection and analysis system. In *ICDE*, pages 1273–1276, 2012.
[20] P. Lofgren and A. Goel. Personalized pagerank to a target node. *arXiv:1304.4658*, 2013.
[21] M. Mathioudakis and N. Koudas. Twittermonitor: trend detection over the twitter stream. In *SIGMOD*, pages 1155–1158, 2010.
[22] S. Phuvipadawat and T. Murata. Breaking news detection and tracking in twitter. In *WI-IAT*, pages 120–123, 2010.
[23] M. Quezada, V. Peña-Araya, and B. Poblete. Location-aware model for news events in social media. In *SIGIR*, pages 935–938, 2015.
[24] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *WWW*, pages 851–860, 2010.
[25] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling. Twitterstand: news in tweets. In *GIS*, pages 42–51, 2009.
[26] L. Shou, Z. Wang, K. Chen, and G. Chen. Sumblr: continuous summarization of evolving tweet streams. In *SIGIR*, pages 533–542, 2013.
[27] K. Watanabe, M. Ochi, M. Okabe, and R. Onai. Jasmine: a real-time local-event detection system based on geolocation information propagated to microblogs. In *CIKM*, pages 2541–2544, 2011.
[28] J. Weng and B.-S. Lee. Event detection in twitter. In *ICWSM*, pages 401–408, 2011.
[29] C. Zhang, S. Jiang, Y. Chen, Y. Sun, and J. Han. Fast inbound top-k query for random walk with restart. In *ECML/PKDD*, pages 608–624. 2015.
[30] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: Concepts, methodologies, and applications. *ACM TIST*, 5(3):38:1–38:55, 2014.