Bursty Event Detection Throughout Histories

Conference Paper · December 2018

CITATIONS
0

READS
50

3 authors, including:

Debjyoti Paul
University of Utah
7 PUBLICATIONS **33** CITATIONS

SEE PROFILE

2019 IEEE 35th International Conference on Data Engineering (ICDE)

Bursty Event Detection Throughout Histories

Debjyoti Paul, Yanqing Peng, Feifei Li
School of Computing, University of Utah
Salt Lake City, Utah, USA
{deb,ypeng,lifeifei}@cs.utah.edu

***Abstract*—The widespread use of social media and the active trend of moving towards more web- and mobile-based reporting for traditional media outlets have created an avalanche of information streams. These information streams bring in first-hand reporting on live events to massive crowds in real time as they are happening. It is important to study the phenomenon of *burst* in this context so that end-users can quickly identify important events that are emerging and developing in their early stages. In this paper, we investigate the problem of *bursty event detection* where we define *burst* as the acceleration over *the incoming rate of an event mentioning*. Existing works focus on the detection of current trending events, but it is important to be able to go back in time and explore bursty events throughout the history, while *without the needs of storing and traversing the entire information stream from the past*. We present a succinct probabilistic data structure and its associated query strategy to find bursty events at any time instance for the entire history. Extensive empirical results on real event streams have demonstrated the effectiveness of our approach.**

***Index Terms*—data streams, bursty event detection, data sketch, synopsis, data summaries**

I. Introduction

Social media has changed how we understand the world today. Every minute more than a billion of micro documents (text, photos, videos etc.) are generated in a streaming fashion. These information streams contain events happening around the world, and have become an important source for us to know the latest local and global events (big or small). Many possible ways exist for classifying or associating a social media record to a particular event [1], [2], [3], [4], [5], e.g., using its hashtag; and it is possible that the content of one record is associated with multiple events. We denote the mentioning of an event e in a social media record as *an event mentioning* for e. The total event mentioning of e in a given time period gives the frequency of e in that period.

Events that are rising sharply in frequency are called *bursty events*. There are many different ways of capturing the phenomena of *burst*. The most intuitive approach is to model *bursts* through the *acceleration of event frequency over time*. Intuitively, the change in frequency of an event over time forms a time series, and the acceleration of that time series at time t is given by the first order derivative of the underlying frequency curve. For a given event e, its acceleration naturally reflects how sharp the aggregated amount of event mentionings (its frequency) has changed for e (in other words, *attention to* e) from the underlying information streams.

Note that a *bursty event* is distinct from a *frequent event*, and may not necessarily associate with high frequency. For example, *weather report* as an event is clearly a *frequent event* since its frequency is high, but it is most likely *not a bursty event* since its frequency is relatively stable over time. In contrast, *an earthquake outbreak* is clearly not frequent, but once has happened, it is most likely to be very bursty (meaning that there will be a significant uprise in a short period of time in the mentioning of earthquake from the information streams). That said, to capture only those important bursty events, one can impose a frequency threshold when detecting bursty events, i.e., only those bursty events with a reasonable amount of frequency are worth capturing.

Monitoring and analyzing bursty events in social media is an important topic in data science, as it enables users to identify, focus attention and take reactions to important uprising events in their early stages while they are still developing. To that end, most existing works focus on detecting current bursty events in real-time [6], [7], [3], [8], [9].

However, it is also important to be able to travel back in time and analyze bursty events in the past throughout the history, in order to support many important and critical analytical operations. For example, in order to understand how a city's emergency network had responded, operated, and coordinated under an emergency event (e.g., a fire breakout, a major accident, etc.), we would like to identify such bursty events in the past and trace how they have developed over time. Such analysis is essential towards enabling deep analysis of historical events and improving the effectiveness and efficiency of future operations and response efforts.

That said, an interesting challenge arises, which is to answer the following types of questions: What are the bursty events in the first week of October in 2016? Is "Anthem Protest" a bursty event in second week of September in 2017? For the first question, one significant bursty event in the first week of October in 2016 is obviously the leak of the "Access Hollywood" tape. For the second question, the answer is most likely no, since the "Anthem Protest" didn't become predominant and gain significant media attention until the 3rd week of September in 2017 as the protests became more widespread when over 200 players sat or kneeled in response to Trump's calling for owners to fire the protesting players on September 24, 2017.

The historical bursty event detection problem have large potential in data mining. Recent work [10] has used the idea of busty event detection in historical data so as to help identifying key events during US election 2016 and their influence to the progress and result of the election. Unfortunately, as far

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00124

as we know that no prior work has been done on historical queries for bursty event detection. Intuitively, a naive approach in answering these queries is to simply store *either the entire information streams*, or *just the frequency curves for all events* over the entire history, which is clearly very expensive. This naive approach is not only expensive in terms of storage, but also in terms of the query cost: traversing a large amount of data through histories is simply not going to be efficient.

Our contributions. To facilitate these queries, an efficient way to detect bursty events within a historical time range is desired. A key observation is that in many of the aforementioned analytical scenarios, *approximate answers* are acceptable, especially when the approximations are provided with quality guarantees and users may tune/adjust the tradeoff between efficiency and approximation quality. This is evident by the significant amount of studies, and their huge success of developing various synopses, sketches, and data summaries for summarizing data streams in the literature [11].

But most of the existing sketching and synopses techniques developed for data streams are designed to *summarize the entire stream up to now*[12], [13], [14]. As a consequence, they are *not able to return the summary for a data stream with respect to an arbitrary historical query time range.*

To the best of our knowledge, no existing techniques support historical burstiness queries. In light of that, this work aims to close this gap and our contributions are:

- We present a formal definition of *burstiness* at any single time instance and formulate the problem of bursty event detection in Section II.
- We design a scheme in Section III, which achieves high space- and query-efficiency, to estimate the burstiness of an event e for any point in history with high accuracy, with respect to a *single event stream* for e.
- We advance the above design in Section IV to a method that is able to estimate the burstiness of *any event* e at any point in history over a stream with *mixture of events*.
- We extend the investigation to *historical bursty time* and *event detection queries* in Section V.
- We present an extensive empirical investigation in Section VI using real event streams to demonstrate the effectiveness of our methods.

Lastly, Section VII presents a review of related work and the paper is concluded in Section VIII with key remarks and future work.

II. Preliminaries

A. Problem Formulation

Let $M = \{(m_1, t_1), (m_2, t_2), (m_3, t_3), \ldots, (m_N, t_N), \cdots\}$ be an information stream of text elements where each message m_i is associated with a timestamp t_i such that $t_i \leq t_j$ iff $i < j$. Here M can be thought of a stream of tweets, microblogs or messages, each of which is represented as a text element/message m_i.

Without loss of generality, we assume that each message m_i discusses *one particular event* from a universal event space Σ in which each event is uniquely identified by a distinct event id, and there exists a hash function h that maps a message m_i to an event id in Σ, i.e., $h : m_i \rightarrow [1, K]$ and $h : m_i \in \mathbb{Z}$ where $K = |\Sigma|$.

For example, the following two messages *"LBC homeboy stoked to see Brasil wins Gold !! @neymarjr"* and *"Featuring @neymarjr #brasil #gold #Olympics2016"* should be mapped to a single event id. Since, both the messages are related to soccer final event at Rio Olympic 2016.

Applying h to stream M, we obtain an event identifier stream (or simply event stream) $S = \{(a_1, t_1), (a_2, t_2), (a_3, t_3), \ldots, (a_N, t_N), \cdots\}$ where $a_i \in [1, K]$ is an event id. It is possible that $a_i = a_j$ for $i \neq j$, and it is also possible that $a_i = a_{i+1}$ and $t_i = t_{i+1}$ which happens when the same event is mentioned by multiple messages with the same timestamp. In the general case, when a message m_i may discuss *multiple events*, we assume that h will map m_i to a set of event ids, and add multiple pairs of (event id, t_i), one for each identified event id, to the event identifier stream S. For example, h can be as simple as using the hashtag of a message m, or a sophisticated topic modeling method like LDA [15], [5], [16] that maps messsage m to a single or multiple topic/s or event/s.

An efficient mapping of messages to event ids is an orthogonal problem statement which can be solved separately. As of now, we consider it as a black box, not to digress from the objective of this paper.

A temporal substream is defined by elements of a stream that falls within a temporal range, i.e., $S[t_1, t_2] = \{(a_i, t_i) \in S | t_i \in [t_1, t_2]\}$. The *frequency of an event e in a given time range* $[t_1, t_2]$, denoted as $f_e(S[t_1, t_2])$, is defined as its number of occurrences in $S[t_1, t_2]$. Finally, we define $F_e(S, t)$ as the *cumulative frequency* of e in stream S up to time t, i.e., $F_e(S, t) = f_e(S[0, t])$. When the context is clear, we simply use $F_e(t)$ to denote $F_e(S, t)$, and $f_e(t_1, t_2)$ to denote $f_e(S[t_1, t_2])$. When t is a parameter, $F_e(t)$ defines a function over the time domain, which we denote as the *frequency curve of e*.

We then define the concept of burstiness based on the incoming rate of an event over a user-defined time span and its acceleration over two consecutive time spans. We use $b_e(t)$ to denote the burstiness of e at time t. Intuitively, if the frequency curve $F_e(t)$ *were* a continuous curve with respect to the time domain, then $b_e(t)$ is simply the acceleration of its frequency curve at t, i.e., $b_e(t) = \frac{d(F_e(t))}{d(t)}$. However, even though time is a continuous domain, in most practical applications, timestamp values are from a discrete domain. Hence, we cannot take the first order derivative of $F_e(t)$ to find its acceleration at t.

To address this issue, we introduce a parameter τ called "the burst span" which captures the time interval length with respect to which we will calculate the burstiness of an event. The *burst frequency* $bf_e(S, t)$ (or simply $bf_e(t)$ when the context is clear) of e at time t with respect to stream S is simply the frequency of e in time range $[t - \tau, t]$, i.e., $bf_e(t) = f_e(t - \tau, t)$ and it measures the *incoming rate* of

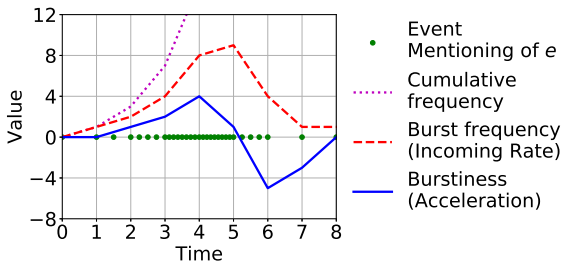


Figure 1: An example of burst where $\tau = 1$.

e in a time span $[t - \tau, t]$.

Definition 1 (Burstiness): The burstiness $b_e(t)$ of an event e at time t is defined by the *acceleration* of e's arrival within two neighboring time spans of length τ each, i.e., $b_e(t) = bf_e(t) - bf_e(t - \tau)$.

Figure 1 gives an example of the burstiness. In this simple example, we set $\tau = 1$. The green dots indicate the occurrences of an event e. Intuitively, the incoming rate is stable in time range $[0, 1)$, then grows quickly in the time range $[1, 4)$. Even though many instances have occurred in the time range $[4, 5)$, but *the increase in incoming rate* is actually slower than that of the previous time span $[3, 4]$. Hence, the burstiness value is 0 in $[0, 1)$, then keeps increasing in $[1, 4)$, and decreases to a smaller value in $[4, 5)$.

It is important to note that burstiness measures the *increase* in incoming rate, rather than the *incoming rate itself*. We are interested in how significant the change of the mentionings is for an event, rather than how frequently an event is mentioned. An event may sustain a high incoming rate in two or more consecutive time spans, but that does not mean it's bursty (if the incoming rates are high but stable). *The larger the burstiness value is, the faster the increase in incoming rate is for two consecutive time spans*. Given a stream S and a time span parameter τ, we define the following queries:

1) POINT QUERY $q(e, t, \tau)$ reports the burstiness of an event e at time t, i.e., $b_e(t)$.
2) BURSTY TIME QUERY $q(e, \theta, \tau)$ returns all timestamps $\{t\}$ such that $b_e(t) \geq \theta$ for a query threshold θ.
3) BURSTY EVENT QUERY $q(t, \theta, \tau)$ returns all events $\{e\}$ such that $b_e(t) \geq \theta$ for a query threshold θ.

Lastly, a special case is when the even stream *contains a single event*, i.e., $S_e = \{(a_i, t_i) | (a_i, t_i) \in S \text{ and } a_i = e\}$. In this case, since all elements contain the same event identifier, we can simplify its representation as $S_e = \{t_i | (a_i, t_i) \in S \text{ and } a_i = e\}$, i.e., it is an ordered sequence of timestamps. S_e may have duplicated timestamp values when an event is mentioned by multiple messages with the same timestamp. A summary of notations is provided in Table I.

B. Baseline Approach

The naive baseline is to store the entire event stream, and then using a simple variation of binary search we can develop solutions for all three query types. The space cost of this approach is $O(n)$ (where n is the size of an event stream), and the query cost of this approach is $O(\log n)$ for a point query, $O(n)$ for a bursty time query if burstiness is not pre-computed and stored and indexed or $O(\log n)$ otherwise, and $O(\log n)$ for a bursty event query. The baseline solution is needed if exact solutions are required, but in practice, the event stream is of extremely large size (as time continues to grow) and this solution becomes expensive. Often time, approximations are acceptable and this gives us the opportunity to explore the approximation quality and efficiency tradeoff.

Table I: Table of Notation

Notation	Meaning
e	an event or an event id
$M = \{(m_1, t_1), (m_2, t_2), \cdots\}$	stream of timestamped messages
$h : m_i \to [1, K]$	map a message to an even id
$S = \{(a_1, t_1), (a_2, t_2), \cdots\}$	event stream with (id, timestamp)
S_e	event stream of e (timestamp only)
$S[t_1, t_2]$	substream of S in time range $[t_1, t_2]$
τ	burst span
$F_e(t)$	cumulative frequency of e in $S[0, t]$
$f_e(t_1, t_2)$	frequency of e in $S[t_1, t_2]$
$bf_e(t) = f_e(t - \tau, t)$	burst frequency (incoming rate) of e at t
$b_e(t) = bf_e(t) - bf_e(t - \tau)$	burstiness (acceleration) of e at t

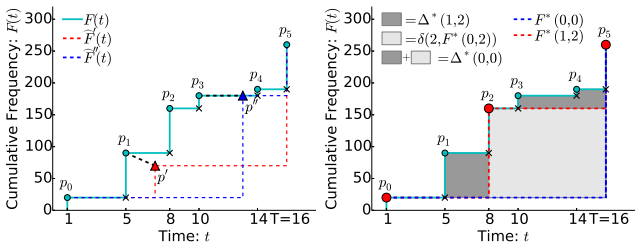
C. Count-Min Sketch

A Count-Min (CM) sketch [12] is a probabilistic data structure that returns an approximation $\widehat{f}(x)$ for the frequency of x ($f(x)$) in a multi-set A, using small space. A CM sketch [12] maintains multiple rows ($O(\log \frac{1}{\delta})$) of counters. Each row has $O(\frac{1}{\varepsilon})$ counters. An incoming element x is hashed to a counter in each row to increment that counter by 1. To estimate the frequency of x, CM sketch returns the smallest counter value among all counters that x is being hashed to from all rows. This guarantees that $\Pr(|\widehat{f}(x) - f(x)| \leq \varepsilon N) \geq 1 - \delta$, where N is the size of the multiset A for some $0 < \varepsilon, \delta < 1$. CM sketch can be easily maintained in a streaming fashion.

III. SINGLE EVENT STREAM

We focus on the POINT QUERY in Sections III and IV, and extend to other query types in Section V. We start with the special case where the event stream contains a single event, i.e., we have S_e for some event e. Under this context, we can drop the suffix e from all notations in the following discussion.

In order to estimate $b(t)$ for any time instance t, one option is to approximate the burst frequency curve (see Figure 1). However, that would require the burst span τ to be a fixed value. We'd like to enable users to query for burstiness while using both τ and t as a query parameter. Hence, we will instead approximate the frequency curve $F(t)$ and show how to use an approximation of the frequency curve to estimate $b(t)$.



(a) Frequency curve and approximation. (b) Bounded error.

Figure 2: PBE-1 example: $T = 16, N = 260, n = 6, \eta = 3$.

To establish an estimator $\widehat{b}(t)$ for $b(t)$, we first make the following observations. By definition,

$$\begin{aligned} b(t) =& bf(t) - bf(t-\tau) \\ =& f(t-\tau, t) - f(t-\tau, t-2\tau) \\ =& (F(t) - F(t-\tau)) - (F(t-\tau) - F(t-2\tau)) \\ =& F(t) - 2F(t-\tau) + F(t-2\tau). \end{aligned} \tag{1}$$

Henceforth, if we have an estimation $\widehat{F}(t)$ for $F(t)$, we obtain an estimator $\widehat{b}(t)$ for $b(t)$:

$$\widehat{b}(t) = \widehat{F}(t) - 2\widehat{F}(t-\tau) + \widehat{F}(t-2\tau). \tag{2}$$

Assume that $\widehat{b}(t)$ and $\widehat{F}(t)$ are random variables, then our estimation error for the burstiness at any time t is also a random variable, which is $|b(t) - \widehat{b}(t)|$. Since users may pose a query at any time instance t, assuming that each time instance is equally likely to be queried, our objective is to minimize the expectation of the cumulative difference between ground truth $b(t)$ and estimated $\hat{b}(t)$, which is simply:

$$\text{minimize} \int_0^T |b(t) - \widehat{b}(t)| \, dt,$$

where integration becomes a summation for a discrete time domain, and T is the latest timestamp in the stream.

To this end, we propose two techniques: one requires buffering of incoming event stream elements, and the other does not and is based on an improvement of Persistent Count-Min sketch (PCM). We dub them PBE-1 and PBE-2 respectively where PBE stands for *persistent burstiness estimation*.

A. PBE-1: *Approximation with Buffering*

Construction. The frequency curve $F(t)$ is a continuous curve as shown in Figure 1 if the time domain is continuous. In practice, most event streams consist of discrete timestamps (as clocks are always discretized to a certain time granularity). In this case, $F(t)$ becomes a monotonically increasing staircase curve as shown in Figure 2a.

Consider an approximate frequency curve $\widehat{F}(t)$ that is *always below* the staircase curve $F(t)$, i.e., $\widehat{F}(t) \leq F(t)$ for all t, and let $\Delta = \int_0^T (F(t) - \widehat{F}(t)) \, dt$. By linearity of random variables, and equations (1) and (2), we arrive at the following objective function:

$$\begin{aligned} F^*(t) &= \text{argmin}_{\widehat{F}(t)} \, \Delta \\ &= \text{argmin}_{\widehat{F}(t)} \int_0^T (F(t) - \widehat{F}(t)) \, dt. \end{aligned} \tag{3}$$

Assume that S has N entries so far up to time T, but they arrive at $n \leq T \leq N$ distinct timestamps, which means that the size of $F(t)$ (denoted as $|F(t)|$) is n. We aim to find an optimal approximation $F^*(t)$ with fewer entries than $F(t)$, that can reproduce $F(t)$ with the *smallest possible errors among all approximations* $\widehat{F}(t)$ *that never exceeds (i.e., never overestimate)* $F(t)$.

Note that we do not have to limit to those approximate curves that are always under $F(t)$ to get a good approximation $\widehat{b}(t)$, but without such limitation the search space for finding a good approximation becomes significantly larger and our results in Section VI demonstrate that the optimal approximation from $\widehat{F}(t)$'s with such a limitation is sufficient to return high quality approximations.

The above approximation error Δ is naturally the difference of the areas enclosed by the approximate and the original frequency curves. And we use Δ^* to represent this error when $\widehat{F}(t) = F^*(t)$. It is easy to show the following lemma by (1), (2), and (3).

Lemma 1: When $\widehat{F}(t) = F^*(t)$, the expectation of $|b(t) - \widehat{b}(t)|$ is minimized among all approximate curves that do not overestimate $F(t)$, and it is at most $4\Delta^*$.

The fact that $F(t)$ is a staircase curve *and* the constraint that $\widehat{F}(t) \leq F(t)$ for any t imply that the optimal approximate frequency curve $F^*(t)$ *must be also a staircase curve*. Formally,

Lemma 2: Under the constraint that $\widehat{F}(t) \leq F(t)$ for any t, $F^*(t)$ must also be a staircase curve.

Proof. We can use either a piece-wise linear curve or a staircase curve to approximate $F(t)$, but $\widehat{F}(t)$ that is a piece-wise linear curve *without any x-axis-parallel line segments* must have some time instances where $\widehat{F}(t) > F(t)$ (when $F(t)$ is still 0). Now, for a piece-wise linear curve $\widehat{F}(t)$ that is: (1) a mixture of x-axis-parallel segments and angular line segments; *and* (2) always below $F(t)$, we can always replace each angular line segment in $\widehat{F}(t)$ with an x-axis-parallel line segment with the same extent on x-axis to reduce the area difference between $F(t)$ and $\widehat{F}(t)$.

For example, consider Figure 2b, $\{p_0, p_1, p_2, p_3, p_4, p_5\}$ are left-upper corner points that makes a non-decreasing staircase curve. Let's say we have selected two points p_0 and p_3 and we have a budget of another one point to introduce to decrease error within p_0 and p_3. Right now if we introduce a point p' anywhere in range $[p_0, p_3]$ and use only non-parallel axis segments, then it will not adhere to constraints $F^*(t) < F(t)$. At least one non-parallel axis segments in $[(p_0, p'), (p', p_3)]$ will cut one of the vertical line segments. But if we only have *axis-parallel segments* to choose and select a point anywhere in range $[p_0, p_3]$, we will decrease the error without violating the constraint.

That said, our problem is reduced to the following: given a non-decreasing staircase curve $F(t)$ with n points (as shown in Figure 2a), find an approximate staircase curve with $\eta < n$ points while minimizing Δ according to (3), i.e. minimizing the area difference between $F(t)$ and $\widehat{F}(t)$, such that $\widehat{F}(t) \leq F(t)$ for any t.

We denote left-upper corner points of any staircase curve $F(t)$ as $P_{F(t)} = \{p_0, \ldots, p_{n-1}\}$. Firstly, we show that, with the constraint $\widehat{F}(t) \leq F(t)$ for any t, the η left-upper corner points that construct the optimal approximate curve $F^*(t)$ must be a subset of $P_{F(t)}$.

Lemma 3: When $\widehat{F}(t) \leq F(t)$ for any t, $P_{F^*(t)} \subset P_{F(t)}$.

Proof. Prove by contradiction. Assume that there $F^*(t)$ has a left-upper corner point $p'_i \notin P_{F(t)}$ where the x-coordinate of p'_i is between the x-coordinates of p_i and $p_{i+1} \in P_{F(t)}$. Note that $F^*(t)$ is always below $F(t)$ for any t. Now if we move p'_i towards the point $p_i \in F(t)$ that will reduce the area enclosed by $F^*(t)$ and $F(t)$. Figure $2a$ shows an example where we move p' to p_1 or p'' to p_3 which leads to smaller error. This contradicts our assumption that $F^*(t)$ has minimized the error Δ with $p'_i \notin P_{F(t)}$. Hence p'_i must be $\in P_{F(t)}$.

Another observation is that the two boundary points in $P_{F(t)}$ must be included in $P_{F^*(t)}$ in order to minimize the error.

Corollary 1: The boundary points of $P_{F(t)}$, i.e., p_0 and p_{n-1}, must be selected by $P_{F^*(t)}$ in order to minimize Δ.

Corollary 1 and Lemma 3 enable us to obtain the optimal approximation $F^*(t)$ with a budget of η points. After including the two boundary points we are left with $(\eta - 2)$ budget and $(n-2) \in P_{F(t)}$ points to choose from. Let $P = \{p_1, \cdots, p_{n-2}\}$ be the rest of points from $P_{F(t)}$, and $F^*(i,j)$ be the *optimal curve* with the smallest approximation error for $F(t)$, among all approximations $\widehat{F}(i,j)$'s formed by using i points chosen from the *first j points in P*, where $0 \leq i \leq j \leq n-2$, and the two boundary points (p_0 and p_{n-1}) that were included for any approximation. Let $\Delta^*(i,j)$ be the approximation error of $F^*(i,j)$.

Let say, we have an approximation $F^*(i-1,x)$ for some x in range $[i-1, j-1]$. Consider a new approximation $F(i,j)$ that is formed by $P_{F^*(i-1,x)} \bigcup \{p_{j+1} \in P\}$, i.e., adding the jth point from P to update the approximate curve $F^*(i-1,x)$ so that it now has i points from the first j points in P (instead of having $i-1$ points from the first x points in P). Let $F(i,j)$'s approximation error be $\Delta(i,j)$.

The reduction in approximation error is exactly the area difference (between an approximate curve and $F(t)$) reduced by $F(i,j)$ compared to that of $F^*(i-1,x)$, and we denote this error reduction as $\delta(j, F^*(i-1,x)) = \Delta^*(i,j) - \Delta^*(i-1,x)$; see Figure 2b. Now we can formulate a recurrence relation as:

$$\Delta^*(i,j) = \min \begin{cases} \min_{x \in [i-1, j-1]} \Delta^*(i-1,x) - \delta(j, F^*(i-1,x)); \\ \min_{x \in [i, j-1]} \Delta^*(i,x). \end{cases}$$

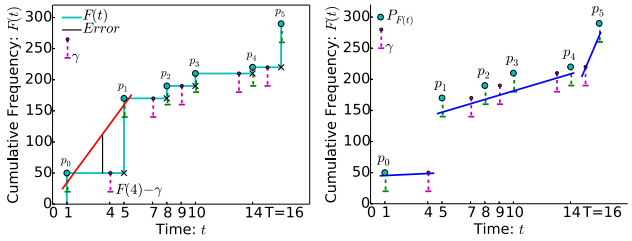
The above recurrence relation can be solved efficiently with a dynamic programming formulation. Our goal is to find $\Delta^*(\eta-2, n-2)$ and the curve $F^*(\eta-2, n-2)$ that corresponds to it.

Algorithm 1: PBE-1 Algorithm (Dynamic Programming)

Input: $P = \{p_0, ..., p_{n-1}\}$ and η
where $p_i = (x, y)$ coordinates ;
Output: P_{F^*} points and min_cost;
// Calculate $\Delta^*(0,0)$;
for $i := 0$ *to* $n-1$ **do**
  $\Delta^*(0,0) = \Delta^*(0,0) + (p_{i+1}.x - p_i.x) \times (p_i.y - p_0.y)$;
end
$min\_cost = \infty$;
$i_{min} = 0$;
for $i := 1$ *to* $n-2$ **do**
  $\Delta^*(i,0) = \infty$;
  for $j := 1$ *to* $min(i, \eta)$ **do**
    $\Delta^*(i,j) = \infty$;
    for $k = j-1$ *to* i **do**
      $\delta = (p_{n-1}.x - p_i.x) \times (p_i.y - p_k.y)$;
      $temp = \Delta^*(k, j-1) - \delta$;
      if $temp < \Delta^*(i,j)$ **then**
        $\Delta^*(i,j) = temp$;
        $bt[i][j] = k$;
      end
    end
  end
  if $i \geq \eta$ *and* $\Delta^*(i,\eta) \leq min\_cost$ **then**
    $min\_cost = \Delta^*(i,\eta)$;
    $i_{min} = i$;
  end
end
Add p_{n-1} to P_{F^*};
$i = i_{min}$; $j = \eta$;
while $i > 0$ **do**
  Add p_i to P_{F^*}; $i = bt[i][j]$; $j = j-1$;
end
Add p_0 to P_{F^*};
return P_{F^*}, min_cost;

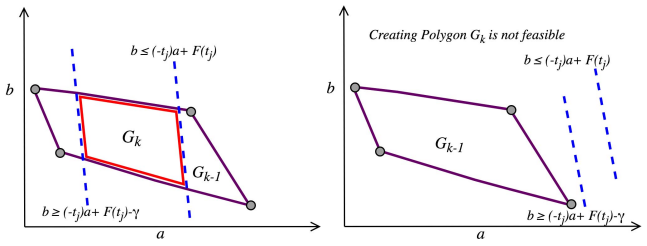
The above discussion leads to an optimal solution to represent an incoming event stream and approximate its frequency curve $F(t)$ with the *smallest* possible error using a space budget of η points. This also leads to the *smallest* possible error for estimating the burstiness for any time instance. The parameter η is a parameter used to trade-off between space and error. An end-user may also impose a hard cap on the error Δ^* instead of imposing a space constraint η. The algorithm above can be easily modified such that it finds the smallest space usage to ensure that a specified error threshold is never crossed.

However, a limitation of this method is that it requires buffering up to n points in $F(t)$ so that it can run a dynamic programming formulation to approximate $F(t)$. Note that in practice, the number of points (n) to represent $F(t)$ could be much less than the actual number of elements N in an event stream S, due to the fact that there could be multiple occurrences of the same event at a given timestamp. That said, PBE-1 maintains $F(t)$ using incoming elements in a streaming fashion, and when $F(t)$ has reached n points for some user-defined parameter n, it runs the above algorithm to approximate $F(t)$ with $F^*(t)$ using η points. It then repeats this process for the next n points in $F(t)$. Parallel processing on mutually exclusive time ranges can be also leveraged to improve system throughput.



(a) Timestamped frequency ranges A. (b) A PLA L for A.

Figure 3: An example of PBE-2.



(a) Polygons G_{k-1} and G_k. (b) Infeasible new constraints.

Figure 4: Online approximation of $F(t)$ without buffering.

Lastly, PBE-1 can also be used as an *offline* algorithm to find the optimal approximation for a massive archived dataset.

Query Execution. Once PBE-1 is constructed, one can easily use it to find the $\hat{b}(t)$ score according to Equation 2. In this equation, $F(x)$ is equal to the y-coordinate of the first point in P_{F^*} before timestamp t. This point can be found quickly via a binary search.

B. PBE-2: *Approximation Without Buffering*

PBE-1 finds a nearly optimal approximation of $b(t)$, but it requires buffering. To remove the buffering requirement, we have to settle for a more coarse approximation. To that end, we develop an online piecewise linear approximation (PLA) technique for approximating $F(t)$ that requires no buffering.

Construction. Unlike the PBE-1 approach where approximation is a subset of points from $F(t)$, a PLA can deviate from points that define the staircase curve of $F(t)$ by a distance (i.e., its approximation error) γ at any time t, where γ is a user-defined error parameter. A too restrictive γ value may result in a lot of line segments in the approximation while allowing more freedom might not be able to capture $F(t)$ accurately.

Recall that the left-upper corner points of $F(t)$ are represented by $P_{F(t)} = \{p_0, p_1, \ldots, p_{n-1}\}$. Each point represents a timestamp and the cumulative frequency at that time instance, i.e., $p_i = (t_i, F(t_i))$; collectively, they define a staircase curve $F(t)$ in the time range $[0, T]$ as shown in Figure 3a.

The basic idea of PBE-2 is to *allow each point on* $F(t)$ *to deviate from* $F(t)$ *by no more than* γ *distance* for any $t \in [0, T]$, i.e, we want to find an approximation $\widehat{F}(t)$ such that $\widehat{F}(t) \in [F(t)-\gamma, F(t)]$ for any $t \in [0, T]$. Note that similar to PBE-1, we also limit our search space to those approximations that do not overestimate $F(t)$.

Now, our problem is reduced to the following problem. Given a sequence of *timestamped frequency ranges* $A = \{(t_0, [F(t_0) - \gamma, F(t_0)]), (t_1, [F(t_1) - \gamma, F(t_1)]), \ldots, (t_n, [F(t_n) - \gamma, F(t_n)])\}$, we want to find a set $L = \{\ell_1, \ldots, \ell_\eta\}$ of piece-wise linear line segments, so that collectively they "cut through" every frequency range in A. Each element ℓ_i in L is a line segment defined by a line $(a_i t + b_i)$ and a time range (t'_{i-1}, t'_i) in which ℓ_i is in effect.

However as shown in Figure 3a, when two neighboring left-upper corner points from $P_{F(t)}$ has a huge gap in their frequency values, say p_0 and p_1, a line segment that cuts through $(t_0, [F(t_0) - \gamma, F(t_0)])$ and $(t_1, [F(t_1) - \gamma, F(t_1)])$ may introduce significant errors (anywhere between 0 and $(F(t_1) - F(t_0))$ when it is used to estimate any frequency value $F(t)$ for some $t \in (t_0, t_1)$.

To address this issue, we introduce additional points to be added to $P_{F(t)}$ to bound such errors. In particular, we require the following points to be added: for every $p_i \in P_{F(t)}$, add $p = (t_i - 1, F(t_i - 1))$ to $P_{F(t)}$. Note that in our definition, *one unit* time is least interval between successive points. In other word, we add the point on the leveling part of the staircase curve right before the staircase curve rises to p_i with a frequency value $F(t_i)$. By the property of a staircase curve and the construction of $P_{F(t)}$, for any $p_{i-1}, p_i \in P_{F(t)}$, $F(t_{i-1}) = F(t_i - 1) < F(t_i)$, and the new $P_{F(t)}$'s size is $2n$. An example is shown in Figure 3a.

Now we derive the set of timestamped frequency ranges A using the newly defined point set $P_{F(t)}$, and search for a piece-wise linear line segments set L that cuts through A. Figure 3b illustrates this process.

A straight line $(at + b)$ that cuts through k *consecutive* timestamped frequency ranges $(t_k, [F(t_k) - \gamma, F(t_k)])$ from A, for some $k = [0, \ldots, 2n - 1]$ is from the set of all (a, b) value pairs that satisfy the following k equations:

$$F(t_j) - \gamma \le at_j + b \le F(t_j), \text{for some } x \in [0, 2n-1], j \in [x, x+k]. \quad (4)$$

We use linear programming to solve the above set of linear equations to solve for valid (a, b) values. Each equation in (4) represents two half-planes as below with parallel edges in the (a, b) space:

$$b \ge (-t_j)a + F(t_j) - \gamma, b \le (-t_j)a + F(t_j). \quad (5)$$

The feasible solutions of (a, b) lie in the region bounded by a polygon G_k created by (5) for k consecutive timestamped frequency ranges from A as shown in Figure 4a.

Note that $P_{F(t)}$ and then its timestamped frequency ranges A can be constructed online in a streaming fashion from S. The following algorithm PBE-2 will find the set of piece-wise linear line segments L as an approximate $\widehat{F}(t)$ to approximate $F(t)$, such that $\widehat{F}(t) \in [F(t) - \gamma, F(t)]$ for any $t \in [0, T]$.

Let G_{k-1} be the polygon created *up to now* with $2(k-1)$ constraints as in (5) from the first $(k-1)$ consecutive timestamped frequency ranges in A. Now, when the next timestamped frequency range (say with a timestamp t_k) from A has arrived, it defines two more constrains as defined by (5). If including these two constraints with the first $2(k-1)$

Algorithm 2: PBE-2 Algorithm

Input: $(t_i, [F(t_i) - \gamma, F(t_i)]), i = 1, 2, \ldots, n$;
Output: L set of lines $\{l_1, l_2 \ldots\}$;
Compute G_2 with
$b \geq (-t_j)a + F(t_j) - \gamma); b \leq (-t_j) + F(t_j))$ for $j \in [1, 2]$;
$i = 1$; $k = 3$;
$start = t_1$;
while $k \leq n$ **do**
  if G_{k-1} *has non-null intersection with* $b \geq (-t_k)a + F(t_k) - \gamma$; *and* $b \leq (-t_k) + F(t_k))$ **then**
    Compute G_k by adding constraints of equations;
    $k = k + 1$;
  end
  else
    Choose (a, b) from G_{k-1} for line $l_i : (at + b)$ for $t \in [start, t_{k-1}]$;
    Add l_i to L;
    $start = t_k$; $i = i + 1$;
    Compute G_{k+1} with $b \geq (-t_j)a + F(t_j) - \gamma)$; and $b \leq (-t_j) + F(t_j))$ for $j \in [k, k + 1]$;
    $k = k + 2$;
  end
end
return L;

constraints can still lead to a bounded polygon G_k, we replace G_{k-1} with G_k and continue (see Figure 4a. Otherwise, we randomly choose a point (a, b) from the area defined by G_{k-1} and add $\ell = (at+b, [0, t_k-1])$ to L; and we throw away G_{k-1} and start constructing a new polygon from scratch. Figure 4b demonstrates an example geometrically.

PBE-2 is able to update its current polygon in $O(1)$ time for each new incoming element (solving two linear equations). To meet a specific space constraint η for maintaining a polygon, we can ask PBE-2 to throw away the current polygon and start constructing a new one whenever there are η vertices to describe the current polygon. Lastly, the following lemma is trivially derived by (1), (2), and the property of $\widehat{F}(t)$ constructed by PBE-2.

Lemma 4: PBE-2 returns an approximation $\widehat{b}(t)$ for $b(t)$ at any time instance that satisfies $|\widehat{b}(t) - b(t)| \leq 4\gamma$.

Query Execution. The query execution of PBE-2 is very similar to the one in PBE-1. We can still use a binary search to find the corresponding segment for any timestamp, and use it to calculate the corresponding $F(t)$ score.

C. Cost Analysis of PBE-1 and PBE-2

For every batch of an event stream with N' elements (that leads to n' points in their frequency curve $F(t)$), PBE-1's space cost is $O(\eta)$. Then $\kappa = \eta/n' \in (0, 1)$ is the constant factor by which the space cost is reduced, compared to the naive exact solution. The space complexity of PBE-1 is $O(\kappa \cdot n)$ where n is the total size of the frequency curve for the entire event stream. PBE-2's space cost varies on the data distribution, as it depends on the size of L at the end of the day when the entire event stream is processed. In both cases of real data sets, the space requirement is much less than that as confirmed by our empirical results in Section VI. The query complexity for both PBE-1 and PBE-2 is the cost of a binary search over the timestamp values, which leads to a time complexity of $O(\log n)$ in the worst case.

IV. Multiple/Mixed Events Stream

In the general case, an event stream S may have multiple event ids. A naive solution is to apply PBE-1 or PBE-2 for each of the single even streams from S. But this means that we would have to maintain up to $K = |\Sigma|$ PBE-1 or PBE-2 structures in the worst case, one for each single event stream S_e for any e appeared in S and in the worst case all events from Σ had appeared in S.

To address this issue, we combine a Count-Min (CM) sketch structure [12] with a PBE construction of same type (PBE-1 or PBE-2).

Construction. We maintain $d = O(\log \frac{1}{\delta})$ rows of cells and each row has $w = O(\frac{1}{\varepsilon})$ cells, where $0 < \varepsilon, \delta < 1$. But instead of using a simple counter as what CM sketch does, we use a PBE at each cell. We also use h_d independent hash functions such that $h_i : e \to [w]$ uniformly for $i \in [1, d]$. Let $\text{PBE}_{i,j}$ represent the PBE at cell (i, j) (jth column at ith row). An incoming element (e, t) is hashed to a cell in each row independently. Once there, we ignore the event id, and treat all elements that were mapped to this cell *as if they were a single event stream by ignoring the potential collisions of different even ids*. We use this single event stream to update the PBE at that cell; see Figure 5 for an illustration. We dub this method CM-PBE (with two variations: CM-PBE-1 and CM-PBE-2).

The query algorithm and the error analysis of the two variants are identical; they only differ in the final error bounds due to the difference in error bounds inherited from PBE-1 and PBE-2. Hence, we simply denote our structure as CM-PBE. Similarly as that in Section III, we first show how to return an approximation $\widehat{F}_e(t)$ for $F_e(t)$ for an event id e and a time instance t using a CM-PBE.

Query algorithm and error analysis. Given a query $q(e, t)$ that asks for $F_e(t)$, we probe each row in CM-PBE using $h_d(e)$'s, and return the estimation $\widehat{F}_{i,h_i(e)}(t)$ using the $\text{PBE}_{i,h_i(e)}$ at cell $(i, h_i(e))$ for $i \in [1, d]$. Note that $\widehat{F}_{i,h_i(e)}(t)$ is an approximation of $F_x(t)$ over a single event stream $S_x = \{(a, t) \in S | h_i(a) = h_i(e)\}$. A key observation is that $F_x(t) = \sum_a F_a(t)$ for all such a's where $h_i(a) = h_i(e)$, which means that $F_x(t) \geq F_e(t)$. So $\widehat{F}_{i,h_i(e)}(t)$ returned by $\text{PBE}_{i,h_i(e)}$, which approximates $F_x(t)$, may overestimate $F_e(t)$. But since each PBE by our construction always return an estimation that is no larger than $F_x(t)$, this will compensate the overestimation on $F_e(t)$.

Lastly, we return the *median* from the d estimations from the d rows as the final estimation $\widehat{F}_e(t)$ for $F_e(t)$. Now, viewing each PBE at any cell as a *black-box counter* for approximating the cumulative frequency $F_e(t)$ for any item e hashed into that cell, the above process becomes similar to that of a CM-sketch where each cell maintains a standard counter to approximate the frequency of any item that is hashed into that cell. That said, using the standard *median-average* analysis,

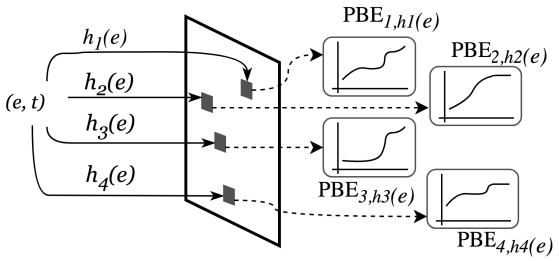


Figure 5: CM-PBE.

the Chebyshev inequality as well as the Chernoff inequality, we can easily show the following results:

Theorem 1: CM-PBE-1 returns an estimation $\widehat{F}_e(t)$ for $F_e(t)$ for any $e \in \Sigma$ and $t \in [0, T]$ such that $Pr[|\widehat{F}_e(t) - F_e(t)| \leq \varepsilon n + \Delta^*] \geq 1 - \delta$. where $0 < \varepsilon, \delta < 1$, Σ is event set, n is the size of $F_x(t)$ for any cell. The same result hold for CM-PBE-2 by replacing Δ^* with γ, where Δ^* and γ are defined in Sections III-A and III-B respectively,

Theorem 1 follows from the linearity of the Count-Min Sketch[12], its own ε-guarantee as we have mentioned it in Section II-C.

A similar analysis as that in Section III will lead to:

Lemma 5: CM-PBE-1 returns an estimation $\widehat{b}_e(t)$ for $b_e(t)$ for any $e \in \Sigma$ and $t \in [0, T]$ such that $Pr[|\widehat{b}_e(t) - b_e(t)| \leq \varepsilon n + 4\Delta^*] \geq 1 - \delta$. The same result hold for CM-PBE-2 by replacing Δ^* with γ.

Space cost. In worst case, for every Δ^* points, PBE-1 generates a segment, resulting in a total space of $O((\frac{N}{\Delta^*} + \frac{1}{\varepsilon}) \log \frac{1}{\delta})$ for CM-PBE-1. However, theoretical justification from random stream model in adversarial condition suggests total space cost is $O((\frac{N}{\Delta^{*2}} + \frac{1}{\varepsilon}) \log \frac{1}{\delta})$, which is also validated with our exhaustive empirical experiments [17]. The Δ-factor improvement on the space is significant as Δ^* is an additive error controlled by user. The same result applies for CM-PBE-2 by replacing Δ^* with γ .

V. Extension

Bursty time query. A bursty time query $q(e, \theta, \tau)$ finds all time instances t's such that $b_e(t) \geq \theta$. Now, consider the single event stream case, a key observation is the incoming rate of e within a linear line segment on its frequency curve $F(t)$ is a constant! Hence, its acceleration (i.e., burstiness) is 0 for that linear line segment. And this is still true for a linear line segment on an approximate frequency curve $\widehat{F}(t)$, regardless whether it is a staircase curve in PBE-1 or a PLA curve in PBE-2. This implies that we only need to ask a POINT QUERY $q(e, t, \tau)$ to PBE-1 or PBE-2 at each time instance when a new line segment starts. In other word, the query cost is linear to the size of PBE-1 or PBE-2 to answer $q(e, \theta, \tau)$.

To answer $q(e, \theta, \tau)$ for a multiple events stream using CM-PBE, we simply carry about the above process for each cell at every row that e is mapped to. We omit the details in the interest of space.

Bursty event query. A bursty event query $q(t, \theta, \tau)$ finds

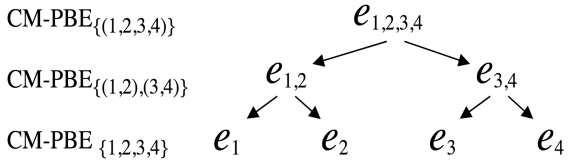


Figure 6: Binary decomposition of the event id space.

events e's such that $b_e(t) \geq \theta$. A baseline approach is query each event id $e \in \Sigma$ using a POINT QUERY $q(e, t, \tau)$. However, this clearly becomes expensive if $K = |\Sigma|$ is large. A minor optimization is to keep the set of event ids that appeared in S and only query those, but still this can be expensive if S contains many distinct event ids.

To find a more efficient and scalable solution is not trivial in this case. Our idea is to build a *dyadic decomposition over the event id space*. We then use a binary tree over these dyadic ranges and a CM-PBE for each level of the tree.

More specifically, consider the example in Figure 6 with only 4 event ids. In the leaf level, we have an event stream S contains mentionings of 4 events $\{e_1, e_2, e_3, e_4\}$ over time, and a CM-PBE$_0$ over is maintained over S using the event id space $\{1, 2, 3, 4\}$. In the second level, we have an event stream S' with *2 events* $\{e_{1,2}, e_{3,4}\}$ where any $(e_1, t) \in S$ or $(e_2, t) \in S$ adds an element $(e_{1,2}, t) \in S'$, and any $(e_3, t) \in S$ or $(e_4, t) \in S$ adds an element $(e_{3,4}, t) \in S'$. We then maintain a CM-PBE$_1$ over S' using the event id space $\{(1, 2), (3, 4)\}$. Lastly, in the root level, we have an event stream S'' with *only 1 event* $\{e_{1,2,3,4}\}$ where any $(e_{1,2}, t) \in S'$ or $(e_{3,4}, t) \in S'$ adds an element $(e_{1,2,3,4}, t) \in S''$. We then maintain a CM-PBE$_2$ over S'' the event id space $\{(1, 2, 3, 4)\}$.

To derive a pruning condition while searching for bursty events using this structure. Consider the subtree that contains $e_{1,2}$ as the parent node and e_1 and e_2 as two children nodes. We have:

$$\begin{aligned} b_1^2(t) =& (F_1(t) - 2F_1(t-\tau) + F_1(t-2\tau))^2, \\ b_2^2(t) =& (F_2(t) - 2F_2(t-\tau) + F_2(t-2\tau))^2, \\ b_{1,2}^2(t) =& (F_{1,2}(t) - 2F_{1,2}(t-\tau) + F_{1,2}(t-2\tau))^2. \end{aligned}$$

A key fact is that $F_{1,2}(t) = F_1(t) + F_2(t)$ for any t by our construction. Hence, we have:

$$b_{1,2}^2(t) = (b_1(t) + b_2(t))^2 = b_1^2(t) + b_2^2(t) + 2b_1(t)b_2(t),$$

which implies that: $b_{1,2}^2(t) - 2b_1(t)b_2(t) = b_1^2(t) + b_2^2(t)$. Hence:

$$\text{if } b_{1,2}^2(t) - 2b_1(t)b_2(t) < \theta^2, \tag{6}$$

then it must be $b_1^2(t) < \theta^2$ *and* $b_2^2(t) < \theta^2$.

Note that at the level with stream S' we can use CM-PBE$_1$ over $\{(1, 2), (3, 4)\}$ to estimate $\widehat{b}_{1,2}(t)$, and CM-PBE$_0$ over $\{1, 2, 3, 4\}$ from the level below for stream S to estimate $\widehat{b}_1(t)$ and $\widehat{b}_2(t)$. Hence, we can easily check (6) to see if we need to go down to the next level of this subtree or not. It is easy to generalize this filtering condition to every node of the binary tree at every level over these dyadic ranges to prune the search

Algorithm 3: Bursty event query

Input: K, t, θ, τ;
Output: E set of events $\{e_i \in K : b_E(t) \geq \theta\}$;
Construct CM-PBE's as described in Section V;
return $recursion(\log |K|, 1, |K|, t, \theta, \tau)$;
Function *recursion*$(lv, l, r, t, \theta, \tau)$**:**
  if $lv = 0$ **then**
    if CM-PBE$_0$.*PointQuery*$(x, t, \tau) \geq \theta$ **then**
      return $\{x\}$;
    end
  end
  else
    $m = \lfloor (l + r)/2 \rfloor$;
    b_p =CM-PBE$_{lv}$.PointQuery$(e_{l,\ldots,r}, t, \tau)$;
    b_l =CM-PBE$_{lv-1}$.PointQuery$(e_{l,\ldots,m}, t, \tau)$;
    b_r =CM-PBE$_{lv-1}$.PointQuery$(e_{m+1,\ldots,r}, t, \tau)$;
    if $b_p^2 - 2b_{l,\ldots,m}b_{m+1,\ldots,r} \geq \theta^2$ **then**
      Result-L = recursion$(lv - 1, l, m, t, \theta, \tau)$;
      Result-R = recursion$(lv - 1, m + 1, r, t, \theta, \tau)$;
      return Result-L $\cup$ Result-R;
    end
  end
  return $\emptyset$;
end

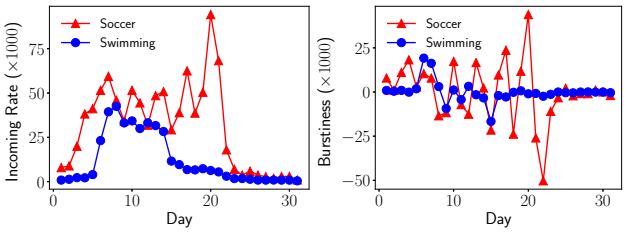
space quickly while searching for bursty events, given a bursty event query $q(t, \theta, \tau)$. As a last step, we remove those from the returned candidates with $b_e(t) < -\theta$. The space cost of this approach is $O(\log K|\text{PBE}|)$, as we have at most $\log K$ levels and each level uses just one PBE. The query cost can be as efficient as just $O(\log K)$ POINT QUERIES (the worst case query cost is still $O(K)$ POINT QUERIES when all events are bursty at t, which rarely happens. In that case, any algorithm has to pay at least $O(K)$ POINT QUERIES costs anyway).

VI. Experiments

In this section, we conduct an experimental study on real-world data sets to evaluate the proposed methods. They are evaluated primarily on three measures: the storage space, the execution (construction and query) time, and the accuracy of the query results.

Data sets. We construct the first data set (olympicrio) by sampling from Twitter data in August 2016 about Olympic Games Rio. We sampled $N = 50,302,975$ tweets in total. All tweets in this data set was given an event identifier based on the type of tweet. We classify the event id of tweets based on hashtags and keywords. We have found $K = 864$ identifiers. We extract the $(timestamp, identifier)$ pair from the data set. The timestamps have a granularity of 1 second, and thus its upper bound is $T = 2,678,400$.

We extract two sub-datasets from olympicrio: soccer and swimming. Figure 7 demonstrates the characteristics of them. Since swimming matches were concentrated in a few days in the first half of the game, we can see a time range of large burstiness over there; after which, both its incoming rate and burstiness decrease to almost zero. On the other hand, soccer matches were held throughout the game, so there are several bursts. The largest burst happens right before the final. To



(a) Incoming rate. (b) Burstiness.

Figure 7: Two events in olympicrio. $\tau = 86,400$ seconds (1 day).

make fair comparison, we then normalize the volume of both datasets to 1 million tweets.

Our second data set uspolitics is also from a sample of Twitter data and we used tweets from June 2016 to November 2016 and focused on events related to US politics. The original data set has 286 million tweets on US politics (e.g. including various events from Election 2016). We found $K = 1,689$ identifiers of events from this data set, which is almost twice than olympicrio. We uniformly sampled 50 million tweets from the original data set to perform a comparative study with olympicrio in section VI-C. Later, we present the trend of uspolitics with full data set. This data set has many events with short period of bursts that can be observed in Figure 13 with intermittent spikes.

Setup. We conducted all experiments on a machine with Intel i7 6th Generation Processor powered with Ubuntu 14.04 LTS. Since a bursty time query is simply a linear (linear to the number of line segments in a PBE) number of point queries, we focus on the investigation of point and bursty event queries.

For a point query, the *approximation errors* of our methods (for both PBE-1 and PBE-2) are additive, which is $|\widehat{b}_e(t) - b_e(t)|$. For a bursty event query, we measure the approximation quality using the standard metrics of precision and recall. *For all approximation error and query time results, we report the average over 100 random queries.*

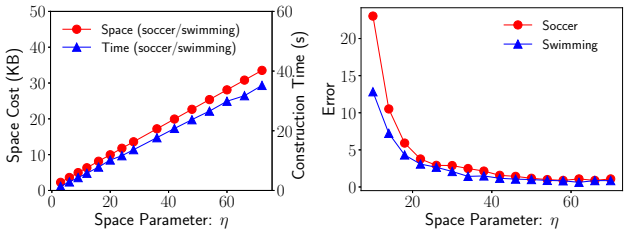
Let n' be the buffer size for PBE-1 where n' is the number of points in the exact staircase curve $F(t)$ of current buffer, i.e., we maintain $F(t)$ for incoming elements and whenever $F(t)$ has reached n' points, we declare the end of the current buffer. Unless otherwise specified, $n' = 1,500$ for all experiments involving PBE-1.

The *baseline method* that stores $F(t)$ exactly for the entire olympicrio or uspolitics requires approximately 10GB.

A. Parameter Study

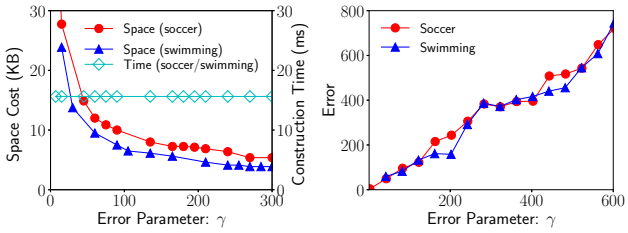
PBE-1 requires a parameter η which controls its space budget for each buffering period; whereas PBE-2 requires a parameter γ which controls the approximation error. We first investigate the impacts of these parameters for PBE-1 and PBE-2 respectively, using *point query over a single event stream*. The event streams tested in this experiment are the soccer and swimming events that illustrated in Figure 7.

Figure 8a shows that as we increase η, which is the size of the approximate staircase curve $F^*(t)$ in each buffer, the



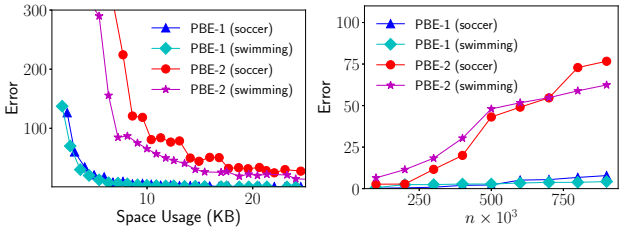
(a) Space and construction costs. (b) Query accuracy.

Figure 8: PBE-1 parameter study.



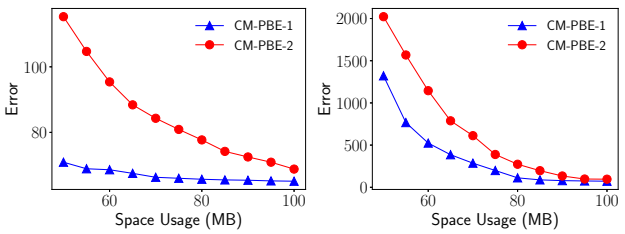
(a) Space and construction costs. (b) Query accuracy.

Figure 9: PBE-2 parameter study.



(a) space vs accuracy. (b) n vs accuracy,|PBE| $= 10$KB.

Figure 10: PBE: single event stream.



(a) olympicrio dataset. (b) uspolitics dataset.

Figure 11: CM-PBE: Space vs accuracy.

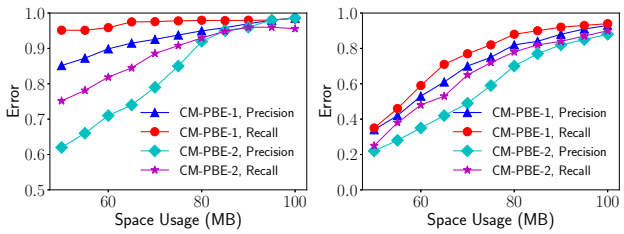
over size of PBE-1 increases linearly as expected. But its overall size is still less than 35KB for the entire stream when η increases to 70. The same figure shows that its construction time also increases with η, as the cost of the dynamic programming formulation for each buffer depends on η. But even in the worst case with $\eta = 70$, its *total construction time from all buffers* for a *one-month* stream is only about 34 seconds. We see similar trend in both soccer and swimming data set, inspite of having different trend in burstiness score (as shown in Figure 7b). In terms of approximation error, PBE-1's approximation quality very quickly with a small increase of η as shown in Figure 8b: when $\eta > 12$, its approximation error is less than 10 for burstiness values that can be as high as more than $25,000$ (see Figure 7b).

Figure 9 reports the impact of the parameter γ to PBE-2 using the same data set and set of queries. When γ is small, such as $\gamma = 2$, PBE-2 takes around 100KB for soccer and swimming data set. As we increase the value of γ, we are tolerating more approximation error at each point. Therefore, the space cost goes down very quickly when γ starts to increase as seen in Figure 9a: the soccer data set uses less than 18KB for the entire stream when $\gamma > 50$. Similarly swimming data set uses around 12KB space when $\gamma > 50$. When γ is large enough, PBE-2 stores only enough information to approximate large bursts and ignore all small fluctuates, and increasing γ won't reduce the space cost significantly anymore. Its construction time reduces somewhat for larger γ values, but is mostly flat and very efficient (solving a few linear constraints and checking half-plane and polygon intersections). Its total construction cost for the entire stream only accumulates to less than 0.016 second as shown in Figure 9a. Figure 9b shows that its approximation error is linear to and bounded by γ (and in practice it is much less than the theoretical bound 4γ), which is as expected.

The above results demonstrate that both PBE-1 and PBE-2 achieve excellent approximation quality (errors in the range of 10s for burstiness values as large as more than $25,000$) using very small space (a few KB), and both of them can be constructed very efficiently.

B. Single Event Stream

In this set of experiments we compare the performance of PBE-1 and PBE-2 under the *single event stream* setting. To make a fair comparison, we adjust and vary their η and γ parameters respectively so that the resulting PBE-1 and PBE-2 use roughly the same amount of space in byte. Figure 10a shows that they yield good approximation errors when given the same amount of the space, while PBE-1 always enjoys a better approximation quality. Next, by fixing the space usage to 10KB for both PBE's, we study the impact of $n = |F(t)|$: number of points to represent the exact curve $F(t)$ for the entire stream. We produce streams of varying length using soccer and swimming so that their n values varies from $100,000$ to $900,000$. Figure 10b shows that both PBE's produce larger errors while n increases, as there are more information in the curve $F(t)$ to summarize while using the same space. Interestingly, the error increases when the incoming rate shown in Figure 7a has a significant change. The reason is that within the range when the incoming rate is stable, increasing n will include the events with a close value of incoming rate than the current last recorded incoming rate. Therefore, it's likely that the new points can be represented by the last staircase or PLA. As a result, the error won't change a lot with the same size. On the other hand, within the range that incoming rate varies, extra staircases or PLAs have to be included in the PBE's to represent the new values. With a fixed size, the error has to increase. This explains the trend observed in Figure 10b.



(a) olympicrio dataset. (b) uspolitics dataset.

Figure 12: Bursty Event Detection: Space vs precision/recall

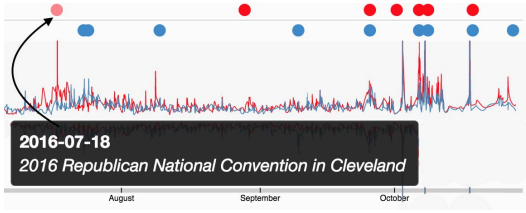


Figure 13: Bursty events from uspolitics and their burstiness values.

C. Multiple Events Stream

For a stream with multiple events, we will use a CM-PBE and we set $\varepsilon = 0.005$ and $\delta = 0.02$, i.e., it introduces an (additional) additive error εn and a failure probability of 2% *in the worst case* to the error introduced by the corresponding PBE's. We first used the entire olympicrio in this set of experiments and studied the performance of CM-PBE-1 and CM-PBE-2. We use a similar set up as that in the single event experiments, where we first allocate the same amount of space (in bytes) to both PBE's (we adjusted their parameters η and γ accordingly to achieve this), and study their approximation quality in Figure 11a. The results show that they still enjoy a high quality approximation error (less than 90 for a burstiness value range that can be as high as more than 25,000), and with a space usage of about 70-100 MB. Note that this is still much better than storing the entire stream. We then run the same experiment on the uspolitics dataset, and the results are shown in Figure 11b. Surprisingly, although these two datasets have the same total volume, the performance on uspolitics suffers more with small spaces. The reason is that events in uspolitics have very different population. Some events attract a lot of attention, while others have only a few discussion. With a small space, the fluctuation of incoming rates of unpopular events are likely to be ignored, and the error can be large. When we increase the space budget, CM-PBE will be affordable to store the trends with finer granularity, and finally lead to good results for both major and minor events.

D. Bursty Event Detection

Finally, we investigate the performance of our method, as described in Section V, for bursty event detection. Our method uses a binary decomposition over the event id space and builds a CM-PBE for each level where the number of events reduces by half as one goes one level above in the tree. This method allows effective pruning (of subtrees) while searching for bursty events with a top-down strategy from the root. In practice, in most cases we only need to issue $O(\log K)$ point queries, roughly $O(1)$ per level, to find bursty events rather than issue $O(K)$ point queries using a naive approach (one per event). This is because in practice, only a small fraction of events are bursty at any given time, so most subtrees will be pruned away using our search strategy and pruning bound.

Our search strategy and pruning bound itself does not introduce any additional errors, but a point query from a CM-PBE at each level may return a two-sided error while approximating $b_e^2(t)$. Hence, Figure 12 investigates the precision and recall of this method (CM-PBE-1 and CM-PBE-2 give similar performance with CM-PBE-1 returning better results) and how they change when we vary the total space usage. For each query, we generated a set of burstiness threshold θ from the range of possible burstiness values of the underlying stream. The results demonstrate that our method is able to achieve high precision and recall using small space. Note that generally the recalls are better than the precisions. This is because when an event is bursting, the incoming rate will have significant change and is likely to be captured by CM-PBE and detected as a bursty event. On the other hand, a bursty event reported can actually be the result of conflicting non-bursty events, so there will be a few false positives in this case. Besides, the results of olympicrio are better than the results of uspolitics, which aligns with the discussion in previous experiments.

Lastly, we present the bursty event trend of uspolitics election 2016 data set. We categorized our events into two category: Democrats and Republican based on its affiliation towards one party. The result of our method shows Democrats and Republican events in timeline along with their magnitude of burstiness in Figure 13. A web page version of this result can be found at estorm.org, where we have marked major event happenings with circle on top of its *bursts*. As an example, our method successfully detected the burst right around the start of the republican party national convention on July 18.

VII. Related Works

Burst Detection. Kleinberg [18] defined the term *bursty* for events, where it is assumed that inter-event gaps x of event follow a density distribution , and a finite state automaton is proposed to model "burstiness". We model "burstiness" as the acceleration of incoming rates over time rather than assuming a distribution function and a fixed set of states over time.

Zhu *et al.* [19] applied Haar wavelet decomposition as their basis to detect "bursts". Fung *et al.* [20] modeled bursty appearance of texts as binomial distribution. He *et al.* [21] applied discrete Fourier transformation (DFT) to identify bursts. Similar to these works, we also have a windowing parameter, τ i.e. to define a burst span for burstiness calculation.

There are also studies on topic discovery in microblogs, where authors used emerging/peaky/trendy as synonyms [22]. Shamma *et al.* [7] modeled *peaky topics* by normalized term frequency score. Lu *et al.* [23] and Schubert *et al.* [24] defined *trendy topic* with a variant of Moving Average Convergence Divergence prediction method to find trending score. AlSumait *et al.* [15] and Cataldi *et al.* [25] used window based approach with online LDA and aging theory over frequency.

Some of the previous studies focus on real-time detecting bursty events in social media. Cameron *et al.* [26] first introduced a bursty event detection system and showed how it plays an important role in crisis management. Xie *et al.* [6] proposed *TopicSketch* to detect events from Twitter data with short response time. Zhang *et al.* [9] proposed *Geoburst* to discover local (define geographically) bursty events. However, none of these works focus on summarizing the burstiness information into a compact sketch. In the case that the user is going to repeatedly issue historical queries with different parameters, the existing techniques can't be directly applied.

Data sketching. Our study is also related to the topic of sketching, building synopses and other data summaries for massive data, which has a rich literature [11]. We leveraged the design from a CM-sketch [12] to build our construction for handling a multiple events stream. Recent studies [17], [27] have also extended some of the sketches to be persistent so that they can support historical queries. Our constructions of PBE are inspired by these studies.

Articles/messages to topic/event mapping. As discussed in Section II-A, many techniques can be used to map an message to one or more events, such as using hashtag or topic modeling method like LDA [16], or topical word embedding model [28], [29], [30], [31]. Hence, we propose this problem to be orthogonal to our work and needs separate attention.

In our paper, we treat this as black box and open ended for improvement. Separating this task comes with certain advantages. *(a) Extensibility:* The module can be upgraded with different features, e.g. *spam detection, bot message detection* without changing the underlying system of burst detection. *(b) Integration:* Projects from different domain can be integrated easily. For example, scientific projects involving *gamma ray bursts* detection involves many data streams with illumination magnitudes over time. Our technique is suitable of reducing the storage space cost with guaranteed approximation.

VIII. Conclusion

This work investigates historical bursty event detection through the design of succinct data summaries that explore the efficiency and approximation quality tradeoff, where burst is defined as the acceleration of incoming rates of event mentionings for a given event. Ongoing and future works include extending our study with a spatial dimension to find historical bursty local events, and investigating how to merge summaries from multiple streams so that our methods can support geographically distributed streams.

IX. Acknowledgment

Debjyoti Paul, Yanqing Peng, and Feifei Li are supported in part by NSF grants 1443046, 1619287, and 1816149. Feifei Li is also supported in part by NSFC grant 61729202.

References

[1] X. Zhou and L. Chen, "Event detection over twitter social media streams," *The VLDB journal*, vol. 23, no. 3, pp. 381–400, 2014.

[2] C. C. Aggarwal and K. Subbian, "Event detection in social streams," in *SDM*, 2012.

[3] C. Li, A. Sun, and A. Datta, "Twevent: segment-based event detection from tweets," in *CIKM*, 2012, pp. 155–164.

[4] 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*, 2015.

[5] C. Xing, Y. Wang, J. Liu, Y. Huang, and W.-Y. Ma, "Hashtag-based sub-event discovery using mutually generative lda in twitter." in *AAAI*, 2016.

[6] W. Xie, F. Zhu, J. Jiang, E.-P. Lim, and K. Wang, "Topicsketch: Real-time bursty topic detection from twitter," in *ICDE*, 2013, pp. 837–846.

[7] D. A. Shamma, L. Kennedy, and E. F. Churchill, "Peaks and persistence: Modeling the shape of microblog conversations," in *CSCW*, 2011.

[8] C. Zhang, L. Liu, D. Lei, Q. Yuan, H. Zhuang, T. Hanratty, and J. Han, "Triovecevent: Embedding-based online local event detection in geo-tagged tweet streams," in *SIGKDD*, 2017, pp. 595–604.

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

[10] D. Paul, F. Li, M. K. Teja, X. Yu, and R. Frost, "Compass: Spatio temporal sentiment analysis of US election what twitter says!" in *KDD*. ACM, 2017, pp. 1585–1594.

[11] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends in Databases*, vol. 4, no. 1-3, pp. 1–294, 2012.

[12] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[13] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *Journal of Computer and system sciences*, vol. 58, no. 1, pp. 137–147, 1999.

[14] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[15] L. AlSumait, D. Barbará, and C. Domeniconi, "On-line lda: Adaptive topic models for mining text streams with applications to topic detection and tracking," in *ICDE*, 2008, pp. 3–12.

[16] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[17] Z. Wei, G. Luo, K. Yi, X. Du, and J.-R. Wen, "Persistent data sketching," in *SIGMOD*, 2015, pp. 795–810.

[18] J. Kleinberg, "Bursty and hierarchical structure in streams," *DMKD*, vol. 7, no. 4, 2003.

[19] Y. Zhu and D. Shasha, "Efficient elastic burst detection in data streams," in *KDD*, 2003.

[20] G. P. C. Fung, J. X. Yu, P. S. Yu, and H. Lu, "Parameter free bursty events detection in text streams," in *VLDB*, 2005, pp. 181–192.

[21] Q. He, K. Chang, E.-P. Lim, and J. Zhang, "Bursty feature representation for clustering text streams," in *SDM*, 2007, pp. 491–496.

[22] A. Guille, H. Hacid, C. Favre, and D. A. Zighed, "Information diffusion in online social networks: a survey," *SIGMOD*, 2013.

[23] R. Lu and Q. Yang, "Trend analysis of news topics on twitter," *IJMLC*, vol. 2, no. 3, 2012.

[24] E. Schubert, M. Weiler, and H. Kriegel, "Signitrend: scalable detection of emerging topics in textual streams by hashed significance thresholds," in *KDD*, 2014.

[25] M. Cataldi, L. Di Caro, and C. Schifanella, "Emerging topic detection on twitter based on temporal and social terms evaluation," in *MDM*, 2010.

[26] M. A. Cameron, R. Power, B. Robinson, and J. Yin, "Emergency situation awareness from twitter for crisis management," in *WWW*. ACM, 2012, pp. 695–698.

[27] Y. Peng, J. Guo, F. Li, W. Qian, and A. Zhou, "Persistent bloom filter: Membership testing for the entire history," in *SIGMOD*, 2018.

[28] A. El-Kishky, Y. Song, C. Wang, C. R. Voss, and J. Han, "Scalable topical phrase mining from text corpora," *PVLDB*, vol. 8, no. 3, pp. 305–316, 2014.

[29] Y. Liu, Z. Liu, T.-S. Chua, and M. Sun, "Topical word embeddings." in *AAAI*, 2015.

[30] X. Fu, T. Wang, J. Li, C. Yu, and W. Liu, "Improving distributed word representation and topic model by word-topic mixture model," in *ACML*, 2016.

[31] Q. Li, S. Shah, X. Liu, A. Nourbakhsh, and R. Fang, "Tweetsift: Tweet topic classification based on entity knowledge base and topic enhanced word embedding," in *CIKM*, 2016, pp. 2429–2432.