Real Time Contextual Summarization of Highly Dynamic Data Streams

Manoj K Agarwal
Microsoft Bing
Search Technology Center - India
Hyderabad – 500032, India
agarwalm@microsoft.com

Krithi Ramamritham
Dept. of Computer Science and Engineering
IIT – Bombay, India
Mumbai – 400076, India
krithi@cse.iitb.ac.in

ABSTRACT

Microblogging streams typically contain information pertaining to emerging real world events. Due to the rapid pace of messages in these data streams, short message size and many concurrent events, it is often difficult for users to understand the full context behind an arriving message. Hence, users resort to the cumbersome task of sifting through many messages to obtain the full context of the underlying event. To address this problem, we propose a novel notion – Contextual Event Summary Threads – and present a technique to extract highly meaningful yet compact event summary threads, capturing the complete context of events appearing in data stream, in real time. Our technique is unsupervised and automatically identifies different facets of live events in an unfiltered data stream in a scalable way and presents them to the users as evolving event threads. Extensive experiments over real data demonstrate that our technique -- while avoiding per message processing -- can summarize live data streams with high accuracy and produce compact event summary threads. The summary size of each event is dependent only on the underlying information and not on the number of messages pertaining to that event. Our technique is generic and is applicable on any chronologically ordered data stream which can be modeled in a <user: message> framework.

CCS Concepts

• **Information systems → Information retrieval → Retrieval tasks and goals → Summarization**

Keywords

Real Time Search; Data Streams; Event Summarization; Algorithm; Experiments.

1. INTRODUCTION

1.1 Motivation

Unstructured data streams -- sequences of chronologically ordered messages posted by multiple users -- occur in various social media and enterprise domains. For example, on Twitter, with a large user base, messages are posted at a high rate. Twitter is often the first medium to report emerging events [14][18]. An event in a data stream is defined by "*messages, posted by multiple users, in the same context, within a bounded time window*", for example, messages posted by the fans during the course of a football match.

An event can be a real world or an abstract activity, relevant for a group of people. It is only natural that in a fast-moving world, a huge number of events occur concurrently.

There have been many recent attempts [14][15][16] at identifying emerging events in real time over live social media streams. Existing unsupervised approaches identify emerging events as temporally and spatially correlated clusters of keywords over dynamic message streams. The temporally and spatially correlated cluster of keywords forms an 'event-topic'. In order to capture the event-topic, a dynamic graph is constructed using the most recent messages, with a sliding window model. Therefore, nodes in the dynamic graph represent temporally correlated keywords. An edge between two nodes -- representing two keywords -- indicates that messages within the recent sliding window contain both the keywords, representing spatial correlation. Thus, nodes of dense sub-graphs embedded in the dynamic graph represent the keyword clusters with strong spatial and temporal correlations [15][16].

When the tweets posted during the Nairobi terrorist attack [30] are fed to the system described in [15], one of the keyword clusters, i.e., event-topic, discovered contained the keywords:

- **A: UK, #kenya, #westgate, #nairobi"**

Clearly, the keywords are insufficient to describe the underlying event. The same is true of another event-topic:

- **B: was, 69, kofi, among, #ghana, attacks, ghanaian, awoonor, killed, poet, prof., #kenya**

Systems such as [14][15] discover event-topics like **A** and **B**. To better understand an event-topic, users have to search for the relevant messages in the data stream by themselves. This burdens the users with the task of understanding the emerging events manually or to determine if there is a connection between **A** and **B**. The shortcomings of the message search-based approach are:

(1) Keyword search over live data streams is primitive, e.g., Twitter just returns the most recent tweets for a given search query [19]. It is not necessary that the most recent tweets are also the most relevant and informative tweets for the event.

(2) Keyword search can produce an information overload for a fast-moving data stream. Often a large number of tweets are returned by Twitter for a search query [5]. Moreover, search results are continuously updated with recent messages. This poses difficulties for the users to keep pace with the evolving events.

Typically, the rate at which messages are generated is high. Hence, even if a subset of the messages in the query response is informative, it is challenging to identify these messages in real-time. The high rate of message arrival, and the fact that the messages are short often makes it difficult for the users to understand the context of a standalone message. Thus, the first goal of this paper is to discover a minimal set of most informative messages from the message stream, related to an emerging event. These messages represent the complete summary of the event. Furthermore, live real-world events are not just point events – they evolve. Our second goal is, for each discovered event, its summary must be updated every time there is any significant change in the event. Note that the events evolving in real time may comprise several different aspects or facets. When these changes in the event summary are arranged temporally, an event thread results, capturing the complete event context with the passage of time.

Given these needs, we present a novel method to automatically extract the Contextual Event Summary Threads in real-time for events unraveling in an unfiltered and fast moving data stream.

In [31], we presented our methodology to create and update an index over discovered 'Contextual Event Summary Threads' in a highly dynamic data stream in real time and enabled keyword search over these events using it. In this paper, we present our technique to summarize the events and to discover the Contextual Event Summary Threads.

1.2 Contributions

Most real-world events comprise several facets. Therefore, the summary for an event is represented as a Directed Acyclic Graph (DAG) that reveals the way the event has evolved. Each node in an event thread, called sub-event, has an associated event-topic similar to **A** and **B**. An event-topic is summarized using a minimal set of most relevant messages discovered from the data stream.

The event thread in Figure 1 was produced automatically by our system [31] from the tweets sent during '**Nairobi Terrorist Attack' [30]**. There are 13 sub-events in the thread in Figure 1. The summary started with a tweet about a mall being attacked, followed by tweets about action against attackers, rumors, claims and counter claims by authorities and citizens, etc., which were discovered in real time. The event thread describes a meaningful chronological sequence of the event. Figure 1 depicts a part of the event thread discovered over 164K tweets. For each sub-event in the event thread, our method identified an appropriate summary.

We identify most appropriate message(s) in the data stream, describing the event-topic. In Figure 1, sub-event 9 corresponds to **A** and sub-event 5 corresponds to **B**. Note, all the keywords for both the event-topics are present in their summaries.

In summary, our system:

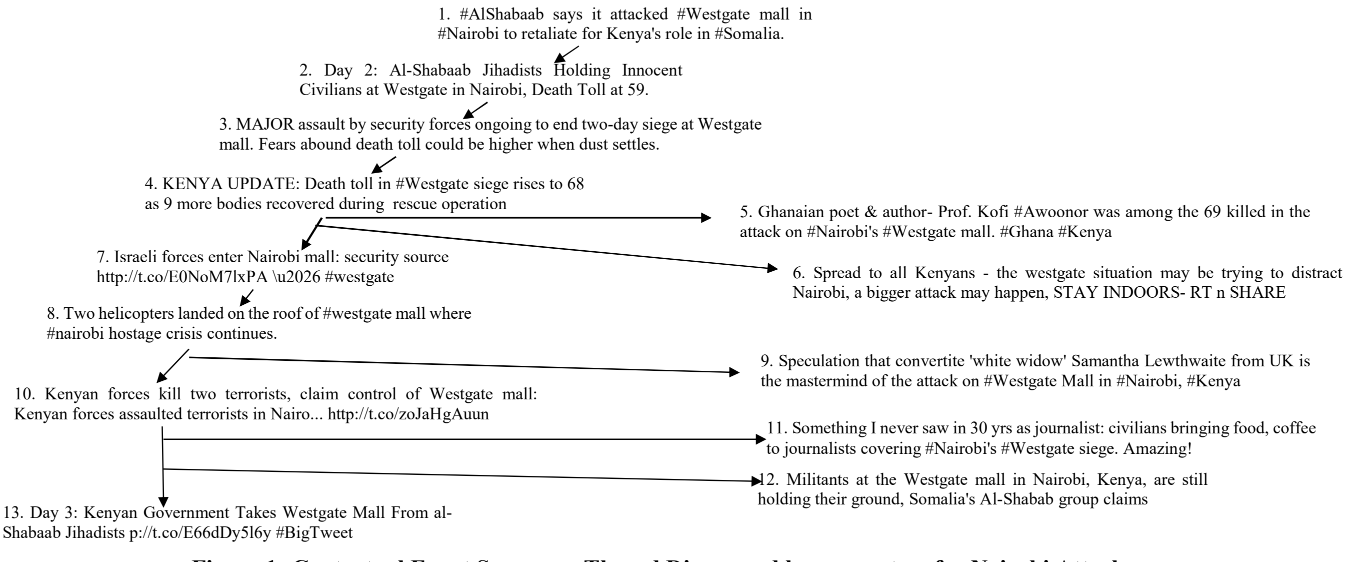
(i) Clusters the related messages together in the data stream: Event-topics are identified by discovering dense sub-graphs, called event-graphs, in the dynamic graph constructed over the message stream [15]. Our system exploits the event-graphs to pool the relevant messages together, related to event-topics.

(ii) Identifies important messages in the message pool to create an event summary: We identify a minimal set of the most relevant messages from the message pool of an event-topic such that the summary is complete and meaningful. In fact, we exploit the structural properties of the underlying event-graph to identify a subset of messages as 'summary candidates'.

(iii) Discovers the event threads for evolving events: As the events evolve, the underlying event-graphs go through structural changes. These changes are tracked in real time. Event summary is updated whenever its event-graph goes through a 'significant' change (cf. Section 6). The updates in the event summaries are captured in contextual summary threads like one shown in Figure 1. Our technique automatically discovers different facets of live events in real time. In general, an event thread can have multiple roots. We say that each unique path in the event thread, from each of its root(s) to each of its leaves, represents a different facet of the event. Event thread in Figure 1 has 6 facets.

In a nutshell, our research contributions involve

1. Summarizing all the events, discovered in a fast-moving unfiltered data stream, in real time, in a scalable and unsupervised manner. For each discovered event-topic, a minimal set of messages is identified to produce a meaningful, informative, stable and complete event summary (Section 5).
2. Tracking the evolution, emergence and dissolution of dense graphs (event-graphs) in a large and highly dynamic graph in the presence of node and edge insertion and deletion.
3. Exposing different facets of live events which are presented as a contextual event summary thread, representing one or more aspects of the event in real time (cf. Section 6).


1. #AlShabaab says it attacked #Westgate mall in #Nairobi to retaliate for Kenya's role in #Somalia.

2. Day 2: Al-Shabaab Jihadists Holding Innocent Civilians at Westgate in Nairobi, Death Toll at 59.

3. MAJOR assault by security forces ongoing to end two-day siege at Westgate mall. Fears abound death toll could be higher when dust settles.

4. KENYA UPDATE: Death toll in #Westgate siege rises to 68 as 9 more bodies recovered during rescue operation

5. Ghanaian poet & author- Prof. Kofi #Awoonor was among the 69 killed in the attack on #Nairobi's #Westgate mall. #Ghana #Kenya

6. Spread to all Kenyans - the westgate situation may be trying to distract Nairobi, a bigger attack may happen, STAY INDOORS- RT n SHARE

7. Israeli forces enter Nairobi mall: security source http://t.co/E0NoM7lxPA \u2026 #westgate

8. Two helicopters landed on the roof of #westgate mall where #nairobi hostage crisis continues.

9. Speculation that convertite 'white widow' Samantha Lewthwaite from UK is the mastermind of the attack on #Westgate Mall in #Nairobi, #Kenya

10. Kenyan forces kill two terrorists, claim control of Westgate mall: Kenyan forces assaulted terrorists in Nairo... http://t.co/zoJaHgAuun

11. Something I never saw in 30 yrs as journalist: civilians bringing food, coffee to journalists covering #Nairobi's #Westgate siege. Amazing!

12. Militants at the Westgate mall in Nairobi, Kenya, are still holding their ground, Somalia's Al-Shabab group claims

13. Day 3: Kenyan Government Takes Westgate Mall From al-Shabaab Jihadists p://t.co/E66dDy5l6y #BigTweet

Figure 1: Contextual Event Summary Thread Discovered by our system for Nairobi Attack

To the best of our knowledge, ours is the first technique that captures the multi-faceted summary of live events and to arrange them in event threads. The event topics, discovered over an unfiltered and fast moving data stream, are summarized in real time in an unsupervised manner, in absence of any user query. The contextual event summary thread is a novel concept in contrast with the story-line generating techniques [4]. Our approach is generic and applicable on any data stream that is a sequence of <user: message>s. Experiments over real data show, our technique leads to compact and meaningful event summary threads for live events.

The organization of the paper is: In Section 2, we present the related work. The event-graph model to capture the state of dynamic data stream and the challenges involved in discovering event summaries are presented in Section 3 and Section 4 respectively. In Section 5 and Section 6, we present the methodology to discover event summary and event summary threads. We present our experimental study in Section 7 followed by conclusion in Section 8.

2. RELATED WORK

Topic Detection and Tracking [1][2] is related to our work. In [12], Yang et al., presented a method to summarize web documents. For these techniques, the document characteristics are completely different from microblog streams ('large size/less in number/offline' vs. 'small size/too many of them/real time'). These methods work on static data only.

In [5], Shou et al., presented a technique to summarize a Twitter data stream, filtered in the context of a given user query. In [23], authors reduce the summarization problem to that of optimizing probabilistic coverage on static data. In [8], Yang et al., present a method to compress the Twitter data stream. Lee et al., in [24] propose a snapshot based method to track the incremental evolution of event-topics. There is significant work [10][11][13] to identify a fixed size summary of microblog posts on a pre-specified topic and on static data, i.e., data which is not updated with new messages.

Gao et al., [3] proposed an unsupervised approach to summarize events, by preparing a joint sentence-tweet level model, across news media and Twitter. Too similar or too different sentence/tweet pair were discarded. In [4], Lin et al. proposed a mechanism to generate a storyline from a microblog data stream. However, the technique is applicable only for static data sets. Vosecky et al., [7] presented a method to identify multiple facets of an event, i.e., important keywords, entities, location, etc., on a static Twitter stream.

The basic difference between existing work and the problem addressed in this paper is: These techniques [1][4][5][7] work in the context of a given query, topic and/or on static data. In contrast, our technique organizes the event summary into event threads, capturing the complete context, for each underlying event discovered in the unfiltered data stream in real time.

The dynamic graph model is exploited to identify the event summary. Our technique exploits the presence of Short Cycle Property (SCP) in the dense sub-graphs of a dynamic graph. In [15], it was shown that dense sub-graphs of a graph possess SCP. There is a large body of work for triangle counting and triangle listing in graphs. Triangle listing is considered an important measure for discovering dense neighborhood [28]. A triangle in a graph induces a cycle of length three. SCP extends this notion to cycle of up to length four, i.e., each node in a graph, possessing short cycle property, participates in a cycle of length at most four within the graph. Triangle listing problem can be divided into processing static graphs [20] and streaming graphs [27]. Streaming graph algorithms for triangle listing are further divided into edge insert graphs [27] and edge insert or delete graphs [9]. In contrast, our algorithm works on a graph with insertion and deletion of both edges and nodes. In this paper, we identify dense event-graphs, discovered based on SCP [15], representing emerging events in a large dynamic graph, with edge and node insert and delete. Our technique keeps track of emergence, evolution and dissolution of the dense sub-graphs in a large dynamic graph. This evolution is presented as contextual event summary thread for the event.

3. EVENT-GRAPH MODEL

Let $S = S_{t-w}S_{t-w+1}\ldots S_t$ represents a message stream. $S_i = \{d^i{}_1 d^i{}_2 \ldots d^i{}_m\}$ is the set of messages arriving in block *i*. A message $d^t{}_j = <u_i, \mathbf{k}>$ in the data stream contains message $\mathbf{k}$ and the *userid* u_j of the user posting the message. A message $\mathbf{k}$ is an ordered set of keywords. *m*, called the *block size*, is the number of messages in a block. The stream is moved forward by expiring the messages in $(t\text{-}w)^{th}$ block and including the messages in $(t+1)^{th}$ block. The message timestamp is implicit with the arrival order in the data stream. For constructing the event thread as shown in Figure 1, we extract the <userid: message> from the incoming data stream. No pre-processing is done on the message, except removing stop-words from it and tokenizing the message into keywords.

Temporal **and** ***Spatial*** **correlation**: The data stream *S* is modeled as dynamic graph $G^t\,(V^t, E^t)$ [15]. For the problem studied in this paper, G^t is an input to our system. The graph $G^t\,(V^t, E^t)$ captures the state of the data stream at the arrival of messages in block *t* ($t \in N^+$). Keywords are represented as nodes in the graph G^t. At time step *t*-1, the graph is updated with the keywords present in the last block of *m* messages and *t*-1 is incremented to *t*. V^t *contains* the *bursty* and *active* keywords in the last *w* blocks at time *t*. Nodes V^t capture *temporal* correlation as only temporally correlated keywords (nodes) are present in the graph.

Table 1: Notation

$S_t; \lvert S_t \rvert = m$	S_t is the block of *m* messages in the t^{th} time step.
$G^t=(V^t, E^t)$	Dynamic graph at time step *t*. V^t represent the keywords and edges E^t the keyword-correlation in the graph G^t.
$G^t{}_c(V^t{}_c, E^t{}_c)$	$G^t{}_c$ is the event-graph embedded in G^t for event *c* at time step *t*. $V^t{}_c$ represent the keywords in event-topic.
$N(v); v \in V^t{}_c$	$N(v)$ represent the set of adjacent nodes to node $v \in V^t{}_c$.
$d^t{}_j=<u_i, k>; 1 \leq j \leq m$	Message $d^t{}_j$ posted by user u_i during time step *t*. $\mathbf{k}$ is set of keywords in the message.
w	Graph G^t contains the bursty and active keywords from last *w* message blocks.
γ	Keyword Burtstiness threshold.
λ	Edge correlation threshold.
$M^t{}_c$	Message set representing summary of event *c* at time *t*.
U_v	Set of userids associated with node $v \in V^t$.
userCover	Set of userids, whose messages include *all* the keywords in an event-topic.

A keyword is *bursty* if it is used in γ (>1) messages in the current window of *m* messages. A keyword *k* is called *active* if its corresponding node $v_k \in V^t$ is present in $G^t\,(V^t, E^t)$, i.e., the keyword is used in at least one of the messages in the last *w* message blocks. An active keyword remains in the graph G^t for as long as it is active. A keyword is removed from the graph G^t if it is inactive for *w* time windows. Hence, a keyword has to be *bursty* at least once to move into the graph G^t. The burstiness constraint helps identify events relevant for a community or group of users.

Each node $v_k \in V^t$ is associated with a user list U_k, containing *userids* of the users who have used the corresponding keyword k since the time it has moved into the graph G^t. This list is used to establish *spatial correlation* among the keywords (nodes). For a pair of nodes $\{v_i, v_j\} \in V^t$, if the similarity score between their respective user lists U_i and U_j is above a given threshold λ, an edge e is placed between the nodes. *Jaccard coefficient (J_c)* is used as the similarity measure. J_c between two nodes v_i, v_j in graph G^t is calculated as $|U_i \cap U_j| / |U_i \cup U_j|$; hence $0 \le \lambda \le 1$. Edge $e \in E^t$ captures the *spatial* correlation between the associated keywords.

Dense sub-graphs in graph G^t represent an event: Dense sub-graphs, embedded in the graph $G^t(V^t, E^t)$ are called event-graphs. The nodes $V^t_c \subset V^t$ in the event- graph $G^t_c(V^t_c, E^t_c)$ are the keywords in the event-topic c and edges $E^t_c \subset E^t$ represent the correlation between these keywords. G^t_c represents the keywords with strong spatial and temporal correlation. In a data stream, modeled as dynamic graph, emerging dense sub-graphs represent the emerging events [15][16].

Short cycles in dense graphs: The length of the Shortest-cycle in a graph has a strong correlation with graph density. For example, triangle listing is a well-known method to measure graph density [27]. Each edge of a triangle induces a cycle of length 3. Similarly, in chordal graphs [29], each node is part of a cycle of length 3. In [15], authors expose a property for the dense sub-graphs embedded in a large graph, called the *short-cycle property*; each node in a dense sub-graph participates in at least one *cycle* of length at most 4, within the sub-graph. Next, we formally define the *Shortest-cycle* and *short-cycle property*.

Def. 3.1.1 *Shortest-cycle*: For a graph $G(V, E)$ and a node $v \in V$, *Shortest-cycle* is the *shortest path P* through which one can return to node v; $\forall e \in P, e \in E$.

Def. 3.1.2 *Short Cycle Property* (*SCP*): For a keyword cluster c, let $G^t_c(V^t_c, E^t_c)$ represent the corresponding subgraph embedded in graph G^t (V^t, E^t) $(\forall k \in c; v_k \in V^t_c)$. $V^t_c \subseteq V^t$, $E^t_c \subseteq E^t$ and $\forall e_{(u,v)} \in E^t_c, \{u,v\} \in V^t_c$. G^t_c possesses the *short-cycle property* if it satisfies the following conditions:

P1. For any two adjacent nodes u and v in the G^t_c, there exists at least *one more* path P between u and v, such that $|P| \le 3$ and $\forall e_{(u,v)} \in P, e_{(u,v)} \in E^t_c$. Therefore, the length of *Shortest-cycle* for $\forall v \in V^t_c$ is ≤ 4. Note, length of the Shortest-cycle for node v is $|P| + 1$ (for edge $e_{(u, v)}$).

P2. For a node $v \in V^t_c$ in graph G^t_c, *all the Shortest-cycles* node v participates in within the graph are of length at most 4.

P3. There is no articulation point in G^t_c. An articulation point is a node whose removal breaks the graph into multiple disconnected components (e.g. in Figure 3(a), node a_4).

In Figure 2 (a), in the absence of P3, clusters C_1 to C_4 will merge into a single cluster since the merged cluster satisfies P1 and P2. Similarly, in the absence of P2, C_1 to C_5 will merge together into a single cluster. Please note, due to SCP, though necessary, it is not sufficient for two event-topics to just share two or more keywords to merge into a single event-topic. In Figure 2(b), C_6 and C_5 share two keywords, but they are not merged together since the merged cluster does not satisfy P2 of SCP. Thus, SCP ensures discovery of dense clusters *efficiently* [15].

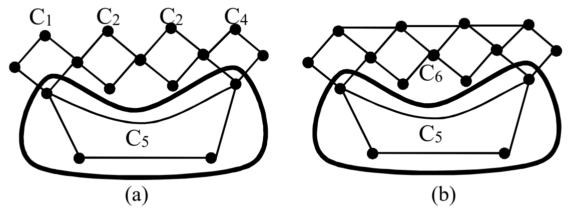


Figure 2: Example event-graphs

Example 1: In Figure 3(a), sets $\{a_1, a_3, a_4\}$ and $\{a_2, a_4, a_5\}$ represent an independent event each. Event-graph in Figure 3(b) represents a single event. In Figure 3(b), keywords having their $J_c \ge 0.25$, have an edge between them. An event-graph satisfies the *SCP* (Def. 3.1.2).

In Figure 3 (b), k_2 and k_3 participate in two *Shortest-cycles* each. Cycle $k_1 \rightarrow k_2 \rightarrow k_4 \rightarrow k_3 \rightarrow k_1$ is an intra-graph cycle too but not the *Shortest-cycle* (Def. 3.1.1). In [15], authors present an efficient algorithm to identify dense-graphs in a dynamic graph by exploiting SCP. SCP ensures that keywords that show a strong temporal and spatial correlation are identified as event-topics.

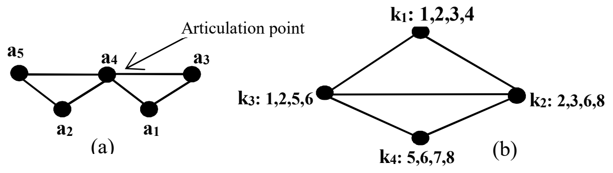


Figure 3: Example event-graphs

4. ISSUES in EVENT SUMMARIZATION

There is a user list U_v associated with each node $v_k \in V^t_c$ in event-graph $G^t_c(V^t_c, E^t_c)$ for event c. v_k is the node corresponding to keyword k. We represent v_k by just v when the context is clear. The messages posted by users in list $U_v, \forall v \in V^t_c$, are considered the pool of message related to event c. Event summary is identified from these messages.

Def. 4.1 Event Summary: Event summary is a *set of messages* $\mathbf{M}^t_c$ from a set of users $U^c \subseteq (\cup U_v; \forall v \in V^t_c)$ such that $\forall v_k \in V^t_c$, $\exists u \in U^c$ where u has used the keyword k in its message(s).

Thus, the event summary is defined as a set of message(s) from a set of users whose messages covers all the keywords in the event-topic. This set of users is also called the valid *userCover*.

Let $u \in U_v$ of the userid list of node $v_k \in V^t_c$. Let $N(v_k)$ represents the set of nodes adjacent to v_k in $G^t_c(V^t_c, E^t_c)$. $\forall n \in N(v_k)$, if $u \in U_n$, node n is considered covered. Thus, all neighbors of node v_k that contain u in their respective user lists are considered covered. Note, *only* the user lists of the neighbors of node v_k in the event-graph are checked for the presence of user u. The messages in the current time window from the users in the *userCover* become part of the event summary $\mathbf{M}^t_c$. The summary $\mathbf{M}^t_c$ for a new event c, emerging in the t^{th} time window is defined as $\mathbf{M}^t_c = \bigcup S^t_u \mid u \in U^c$ where S_u^t is a set of messages from user u in t^{th} message block. $\mathbf{M}^t_c$ is the ordered sequence of messages based on the message timestamp.

Def. 4.3 *Optimal UserCover*: An optimal *userCover T* is the smallest subset of users such that $\forall U_v \mid v \in V^t_c, U_v \cap T \ne \phi$.

Theorem 1: *Discovering optimal userCover is NP-hard.*

Proof Sketch: Hitting set is a well-known NP-complete problem [25]. It is defined as follows: Given a collection S of sets $S_i s$; $1 \leq i \leq n$, find the smallest size set H such that $\forall S_i \in S, S_i \cap H \neq \phi$. By mapping each element in a set S_i to a user id in *userCover,* Hitting set problem is polynomial time reducible to *userCover*. If T* is a solution for *userCover* and H* is a solution for Hitting set, |H*| = |T*|. □

Theorem 1 states, discovering optimal *userCover* is NP-hard even when the user lists associated with keywords are static. However, the set of nodes V^t_c in an event graph G^t_c as well as the user lists U_v associated with each node $v \in V_c^t$ are highly dynamic due to continuous updates in the message stream. Thus, it is non-trivial to identify a minimal set of most relevant messages from the data stream that ensures that the event summary be informative, compact, complete, meaningful and stable.

Identifying users (and not messages) helps us create a more meaningful event summary (at times, users post multiple messages in a small time window to explain the full context of their messages). Since user lists are large for popular events, it is non-trivial to identify optimal *UserCover*. To efficiently identify these users, each arriving message is assigned a score in a scalable manner to rank more relevant messages higher (cf. Section 5.3).

We next highlight how the dynamic updates in the data stream may result in an unstable event summary:

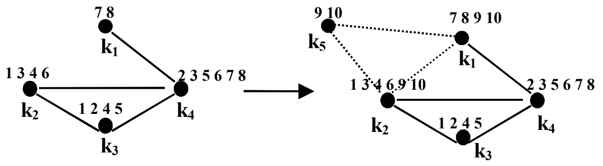


Figure 4: Evolution in the event-graph

Live updates can make a summary unstable: For an event present in a live data stream, let its initial event-graph be as shown in Figure 4(a). The *userCover* had been identified as {1, 3} by any *userCover* discovery algorithm (**k1** in Figure 4(a) is not part of the event-graph). In the next time step, the event-graph gets updated to as shown in Figure 4(b). However, the same algorithm identifies {2, 9} as *userCover*, which results in a different set of messages as summary, making it difficult for users to keep track of evolving event. Thus, due to live updates, event summary may be unstable.

The evolution of the summary for a live event is handled as described in Section 6. We present our algorithm to summarize the event by identifying an approximate *userCover* in Section 5.

5. DISCOVERING EVENT SUMMARIES

In this section, we present our method to discover meaningful event summaries, with the aid of central nodes, called pivot nodes, in the dense event-graphs. In Section 5.2, we present our algorithm to identify the event summaries. In Section 5.3, we present our method to rank the messages in the data stream.

5.1 Pivot Nodes and Pivot Edges

Def. 5.1.1 ***Pivot Edge***: An edge that participates in more than one *Shortest-cycles* within the event-graph is defined as a pivot edge. Higher the number of Shortest-cycles a pivot edge participates in, the more central it is in the event graph.

Def. 5.1.2 ***Pivot Node***: Nodes associated with a pivot edge are called *pivot nodes.*

Nodes which are not pivot nodes are called *peripheral* nodes. In Figure 4(b), edge (**k2**, **k4**) is a pivot edge. {**k2**, **k4**} are pivot nodes.

Lemma 1: *A graph possessing short-cycle property with more than one shortest-cycles within the graph, has a pivot edge.*

Proof: Let event-graph G^t_c (V^t_c, E^t_c) has multiple short cycles, satisfying SCP. Let C (V, E) be a cycle in the event-graph G^t_c such that it has no common edge with any other cycle in the graph. Let $n \in C(V)$ be a node common with another cycle in graph G^t_c (if cycle C has not even one node common with any other cycle within the graph; G^t_c will be disconnected); Since no edge in cycle C participates in another cycle, for any node v in $C(V)$ - n, and for any node u in V^t_c – V, there exist just one path from v to u via node n. Hence in graph G^t_c, node n is an articulation point, violating Def. 3.1.2 of SCP event-graph. Therefore, there must exist another path from v to u. Hence, there exist a cycle $v \to n \to u \to v$. Hence any edge $e \in C(E)$ in path $v \to n$ participates in another cycle, i.e., e is a pivot edge. Therefore, for each Shortest-cycle C (V, E) in an event-graph, $\exists e \in C(E)$ that is a pivot edge. □

Corollary of Lemma 1: *Either a node itself or one of its neighbors is the pivot node in the event-graphs that possesses SCP.* Due to this corollary, we can create a pivot edge cover (*PECover*), as described below.

Def. 5.1.3 PECover: For G^t_c (V^t_c, E^t_c), a *PECover* is a subset of pivot edges $E_p^o \subseteq E^t_c$ such that, $\forall v \in V_c^t$, $\exists e(u,v) \in E_c^t \mid v \in V_p^o$.

Hence, a *PECover* (corresponding *PNCover*) E_p^o (V_p^o) is a subset of pivot edges (corresponding pivot nodes) in the event-graph G^t_c (V^t_c, E^t_c) such that $\forall v \in (V_c^t - V_p^o)$, node v is adjacent to a node in V_p^o .

Let U_v be the user list for node $v \in V^t_c$ and C be a collection of all the user lists in G^t_c (V^t_c, E^t_c). Let C_p be a collection of user lists associated with pivot nodes V_p^o in graph G^t_c ($C_p \subset C$).

Lemma 2: *For collection* C_p $\forall U_i \in C - C_p ; \exists U_j \in C_p \mid$, $U_i \cap U_j \neq \phi$.

Proof: For two nodes v_i and v_j and their associated user lists U_i and U_j in an event-graph G^t_c(V^t_c, E^t_c), if edge (v_i, v_j) $\in E^t_c$, $|U_i \cap U_j| \geq \lambda . |U_i \cup U_j|; \lambda > 0, |U_{l \in \{i,j\}}| \geq \gamma$. Therefore, for any edge $e \in E^t_c$ there exists at least one userid which occurs in the user lists of both the nodes. Since, each node in G^t_c is adjacent to a node in the *PECover* of G^t_c, therefore, for collection C_p, $\forall U_i \in C - C_p ; \exists U_j \in C_p \mid U_i \cap U_j \neq \phi$. □

Hence, we first identify a set of pivot edges, called *PECover.* A valid *userCover* can be identified *only* from the user lists associated with pivot edges (cf. Lemma 2). Thus, pivot edges help us avoid processing a large number of messages associated with *peripheral* nodes and yet ensure that the event summary is complete.

Further motivation to use the *pivot edges* is explained below:

5.1.1 Informative

A message containing more keywords from the event graph is considered more informative. Naturally, a keyword with higher number of neighbors in the event graph is likely to contain such messages. We capture this notion of informative-ness as follows:

Def. 5.1.4 *Informative-ness*: Informative-ness of a node v is defined as $N(v)$, the set of nodes adjacent to v in event graph G^t_c.

Let edge $e\ (u, v)$ be a pivot edge. Let s be a neighbor of u such that $e\ (u, s)$ be a non-pivot edge (s is a *peripheral* node). $N(v)$ represents the set of nodes adjacent to v in $G^t\ (V^t, E^t)$.

Lemma 3: $|N(u)| > |N(s)|$.

Proof: Omitted (cf. Lemma 5). □

Lemma 3 shows that the messages from the users associated with the pivot nodes contain more keywords from the event-topic. Identification of summary from these messages leads to a more informative and compact summary.

5.1.2 Stable

Since a pivot edge participates in multiple cycles, it is more stable than the other edges in a dynamic event-graph. Even if one or more edges/nodes, among edges and nodes adjacent to a pivot edge get deleted in the underlying event-graph, the message set identified as the event summary continues to be a valid event summary (cf. Section 6.2). Hence, identification of event summary based on pivot edges, leads to more stable event summary.

5.1.3 Complete

An event summary containing messages that cover all the keywords in an event-graph is considered a complete event summary. However, we identify the users only from the user lists associated with pivot nodes, i.e., from the collection C_P.

Lemma 4: *A user cover identified only from the pivot nodes of an event-graph, possessing short cycle property, is a valid userCover.*

Proof: The corollary of Lemma 1 along with Lemma 2 completes the proof. □

Lemma 4 shows that the *userCover* identified from the user lists in collection C_P results in a valid *userCover*. With the help of pivot edges, we identify a subset of user lists C_P (from the entire user list collection C for event c) and a valid *userCover* can be identified only *from* these sets. Thus, the event-graph structure helps us identify a small number of more relevant user lists for discovering a valid *userCover*. However, discovering optimal set of pivot nodes in an event-graph remains an NP-hard problem (cf. Theorem 3).

5.1.4 Optimal Summary Size

We next show that the summary size discovered with the aid of optimal *PECover* leads to the optimal size summary.

As shown in Lemma 2, once a pivot edge $e_{(u,v)}$ is identified, all the nodes adjacent to the pivot nodes $\{u, v\}$ can be covered using only the user lists associated with these nodes. We now show that there cannot be any smaller collection of user ids than optimal size *PECover* that results in a valid *userCover*.

Let T* represent the optimal collection of user ids that provide a valid *userCover* for a given event-graph G^t and let *PECover** be the optimal *PECover.*

Theorem 2: $|PECover^*| \leq |T^*|$.

Proof: For a given event-graph $G^t\ (V^t, E^t)$, a dominating set D is subset of nodes in V^t such that each node in $(V^t - D)$ is adjacent to a node in D [25]. Let $D^* \subset V^t$ be the optimal size dominating set. Therefore, D^* is the smallest set of nodes such that each node in $V^t - D^*$ is adjacent to a node in D^*.

Step 1: Each node in V^t, is adjacent to a pivot edge (corollary of Lemma 1). Therefore, for each node v in D^* we get a corresponding pivot edge as follows: either edge $e(v, u)$ is a pivot edge; $u \in N(v)$ or edge $e\ (u, N(u))$ is a pivot edge.

In this manner, we get a pivot edge cover *peCover* from D^* which is a valid PECover since D^* is a dominating set for graph G^t; $|peCover| = |D^*|$. Let *PECover** is the optimal *PECover.* Hence, $|PECover^*| \leq |peCover| \leq |D^*|$.

Step 2: Any two nodes that share an edge, have at least one userid common (Lemma 2). Let T^* be the optimal set of user ids, providing the valid *userCover*. Since D^* is the optimal dominating set, therefore, $|T^*| \geq |D^*|$ because for each node in D^*, at least one userid has to be selected in T^* (since the user lists of *only* the neighbors of a node in the event-graph are considered while constructing *userCover*)

From Step 1 and Step 2, $|PECover^*| \leq |T^*|$ □

Thus, optimal pivot edge cover leads to optimal summary size.

On event graph G^t_c we induce a graph $G'(V', E')$ such that each edge in G' is a pivot edge in G^t_c (with the aid of Lemma 5, Section 5.2). We identify a smallest subset $V_{PN} \subset V'$ of nodes in G' such that every node in $V' - V_{PN}$ is adjacent to at least one node in V_{PN}. We call the set V_{PN} Pivot nodes cover or *PNCover*.

Theorem 3: *Discovering optimal PNCover for a given event-graph* $G^t_c(V^t_c, E^t_c)$ *is NP-hard.*

Proof Sketch: Dominating set is a NP-complete problem [25]. It can be shown that $DominatingSet \leq_P PNCover$. □

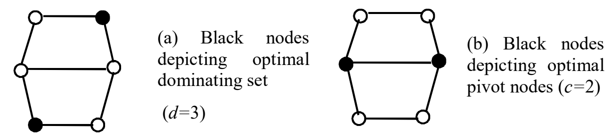

(a) Black nodes depicting optimal dominating set
(d=3)
(b) Black nodes depicting optimal pivot nodes (c=2)

Figure 5: Dominating set Vs. Pivot nodes over event-graph

An alternative to pivot edges is to identify the dominating set of nodes [25] itself in an event-graph. Our primary reason to choose the pivot edges over dominating set is, nodes in dominating set divides a graph into stars whereas the pivot edges divide it into cycles (i.e., quasi-cliques). The *userCover* discovered from nodes that are part of same quasi-clique leads to more informative event summary. For instance, black nodes in Figure 5 (a) and Figure 5 (b) both represent the dominating sets. However, Figure 5 (b) represents the dominating set induced due to the pivot edge. Therefore, *pivot edges* ensure that a message summary discovered based on *pivot edges* is more informative, stable, complete, and compact.

Pivot nodes are 'central nodes' in an event graph. There are other notions of central nodes in a graph, for example, HITS [17] and PageRank [21] identify central nodes (authorities). However, there are basic differences in the settings of our two problems as:

a) [17][21] techniques are iterative in nature, therefore not amenable to rapidly changing dynamic graphs;

b) The notion of 'completeness' (Section 5.1.3) is not applicable for these methods. Due to the same reason, the notion of betweenness centrality [22] is not applicable;

c) These techniques exploit the link structure to identify the authorities, whereas we exploit the graph structure to identify the pivot nodes (since the objectives of two problems are different).

5.2 Approximate User Cover

Next, we present our algorithm to identify approximate *userCover*. On an event-graph $G^t_c(V^t_c, E^t_c)$, we induce a graph $G'(V', E')$, $V' \subseteq V^t_c$ and $E' = E^t_c \bigcap (V' \times V')$ such that $\forall v \in V'$; v is a pivot node in graph G^t_c. The induced graph G' is called the *core* event-graph. The event summary discovered from $G'(V', E')$ follows the same notion of event-summary (Def. 4.1) except that it is discovered on induced graph G'.

Instead of identifying the *userCover* for graph G^t_c, we identify the *userCover* on *G'*. The core event-graph *G'* does not contain the *peripheral nodes* in G^t_c. However, presence of peripheral nodes in the event-graph G^t_c defines the pivot nodes. Therefore, peripheral nodes impact the event summary *only* indirectly.

Lemma 5: A node v with degree $\Delta(v) > 2$ in graph $G^t_c(V^t_c, E^t_c)$ is a pivot node.

Proof: $\forall v \in V^t_c, \Delta(v) \geq 2$ (node v is part of a cycle). Let $v \in V^t_c | \Delta(v) > 2$ be a node in graph G^t_c such that v is not a pivot node. Since $\Delta(v) > 2$, node v is part of more than one cycles. Let C_1 and C_2 be two cycles node v is part of. Let edge $e_{(v,\, n')}$ belong to C_1 and edge $e_{(v,\, n'')}$ belong to C_2. Hence, there exists one path from node n' to node n'' via node v. However, if this is the only path between the two nodes, node v becomes an articulation point violating the short cycle property. Hence there exists one more path p from node n' to n'' inducing a cycle C ($v \rightarrow n' \xrightarrow{p} n'' \rightarrow v$). Length of cycle $|C| \leq 4$ (*SCP*). Therefore, edge $e_{(v,\, n')}$ participates in two cycles C_1 and C. Hence $e_{(v,\, n')}$ is a pivot edge and v is a pivot node. □

With the aid of this lemma, it is easy to induce graph *G'* on G^t_c. Note, *G'* may not follow the *short cycle property*.

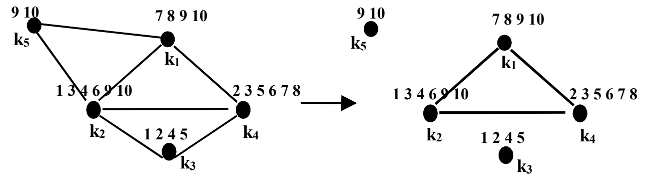


Figure 6: Discovering core graph from an event-graph

Each node v in *G'* is assigned a score $p(v)$; $p(v) \leftarrow d_v(G^t_c)$; $d_v(G^t_c)$ is the degree of node v in graph G^t_c. This score is used to identify *userids* in the *userCover*. In Lemma 2, it is shown that if two nodes are adjacent, they have one or more userids common. Therefore, by selecting the nodes with higher score, event summary is likely to contain more keywords in event-graph G^t_c. We describe our algorithm to find *userCover* for graph *G'* below.

Algorithm `approxUserCover (nodeList nL)`

```
1.  Let $G^t_c(V^t_c, E^t_c)$ be an event-graph.
        a.  let $cId$ be its clusterId;
2.  G' is the core event-graph induced on graph $G^t_c$.
        a.  $\forall v \in G'(V', E'); p_v \leftarrow d_v(G^t_c)$
        b.  $U_v \leftarrow$ list of userids associated with node $v$
3.  $S \leftarrow \Phi$
4.  for $\forall v \in G(V')$ {if $v \notin nL$ {$S \leftarrow S \bigcup v$ }}
5.  cId.nL ← V'; /*we record the pivot node to efficiently update
                    the event summary when cluster evolves later*/
6.  uC ← Φ /*userCover is set to null*/
7.  while ( $S \neq \phi$ )
        a.  $v \leftarrow \arg Max_{v \in V'} (p_v \times d_v(G'))$ /*return node with
                                    highest $p_v \times d_v(G')$ score*/
        b.  $U \leftarrow U_v$ ;
        c.  Let $U_n$ be userids set associated with a node $n$, such that $n$ is
            neighbor of node $v$ in $G'$.
                i.   $T \leftarrow \arg Max_{t \in U \bigcap U_n} messageRank(t)$
                ii.  $uC \leftarrow uC \bigcup \{T\}$;
                iii. $U = U - U \bigcap U_n$ /*We do not select any more userids
                     covering same keyword*/
        d.  For each remaining neighbor, check if userid $T$ exists in their
            userid list
                i.   $A \leftarrow v.adjList(G')$ /*adjacency list of $v$ in G'*/
                ii.  $\forall u | u \in A - n$, if $T \in U_u$, $S \leftarrow S - u$;
8.  return uC;
```

Each edge in graph *G'* is a pivot edge in G^t_c. The degree of a node v in *G'*, $d_v(G')$ is a measure of the number of cycles it participates in graph G^t_c. *messageRank* (.) returns the userid of the message with highest rank. Since, a user is added in the user list of a node one at a time, it is kept sorted in the message score efficiently. In the above algorithm, each node in the event-graph is visited *exactly* once. Thus, the complexity of identifying *userCover is* $O(|V^t_c|)$. Algorithm `approxUserCover` is greedy and achieves the same complexity as Dominating set [25], i.e., (1+ log (Δ)) OPT; OPT is optimal *userCover* size and Δ is the maximal degree in graph *G'*. Dominating set problem is LOG-APX-COMPLETE and no better bound is possible.

5.3 Ranking the Messages

Since our objective is to summarize a highly dynamic data stream in real time, the methodology to rank each arriving message must be i) fast and efficient; ii) rank more meaningful messages higher; and iii) does not re-rank the already ranked messages with fresh updates in the data stream. There are many studies related to ranking the microblogs [18][19][6]. A common conclusion across these studies is that the rank of a tweet depends on the authority of its author and the authority of the message. We exploit these features to establish the rank of a tweet.

Let $d_i, d_j \in S_t$ be two messages in the data stream *S*. Let *R*(.) be a monotonically increasing function such that if d_i is deemed more important than d_j, $R(d_i) > R(d_j)$. To efficiently rank the messages, *R*(.) is applied on both these messages independently.

A tweet is considered more meaningful i) if it is retweeted more (retweet count *RT* captures the authority of the tweet [18]); and ii) if it is tweeted by a person with many followers (follower count *f* captures the authority of the user [19]). Therefore, the tweet score *R*(*d*) of a tweet *d* is computed as:

$$R(d) = \alpha RT.\log f$$

Since the dynamics of an event vary at much finer time scale compared to a user's follower's count, *f* is a logarithmic factor.

The ranking function scores each arriving message efficiently, as soon as it arrives such that the more important messages are likely to be ranked higher. Please note the first criterion to choose a message is the underlying event-graph structure. The highest ranked messages associated with the *pivot nodes* in the event-graph are identified as summary (Section 5.2). Therefore a message from a user with lesser following and/or retweets would be picked in the event summary, if it is more relevant in the context of an event.

In summary, *we translate our goal to provide an event's summary into one of discovering a set of users, collectively using all the keywords in the event-topic. We show, this set of users can be* *discovered only from pivot nodes in the event-graph. We also show that pivot nodes enable us to discover informative, stable and compact summary.*

6. DISCOVERING EVENT THREADS

In this section, we present our technique to discover the contextual event summary threads capturing the evolution of live events. The evolution of an event is tracked by tracking changes in its underlying event-graph $G^t_c(V^t_c, E^t_c)$.

A possible approach for constructing the event threads is to maintain the snapshots of each event-graph in each *time window* [24], but it is not practical to construct summary threads in real time from these snapshots, as 1) instead of incrementally processing the graph, the complete event-graph needs to be processed for each event for each snapshot, thus making it impractical for processing a fast moving data stream in real time; 2) it is shown in [31], that a highly efficient mechanism is needed to keep the indexes updated for the event threads. Thus, it is not practical to create the index again for each snapshot.

Hence, we maintain contextual event threads for each event by keeping a corresponding *eventTree. EventTree* captures the evaluation of the event-graph. We assign a unique event-id (called *clusterId*) to each event-graph. When an event-graph evolves, the summary is updated and the change is recorded in its *eventTree*. With changes in event-graph, *eventTree* is maintained as follows:

Incremental changes in the event-graph: There are incremental changes in the event-graph due to addition and deletion of nodes and edges. Due to these changes, the summary may or may not change but the *clusterId* of the event-graph remains the same.

Disruptive changes in the event-graph: If the structure of an event-graph changes so much that we need to assign it a new *clusterId*, such a change is called disruptive change. For example, when two independent event-graphs with *clusterId* c_1 and c_2 merge into a single event-graph c due to emergence of new nodes/edges. The mapping $c \leftarrow c_1, c_2$ is recorded in the *eventTree.* When two event-graphs merge, their corresponding threads also merge in a single thread. Similarly, due to deletion of nodes/edges, if an event-graph c breaks into say two sub-graphs, c_1, c_2, we record $c_1 \leftarrow c$ and $c_2 \leftarrow c$ in the *eventTree*. Thus, a *eventTree* captures the evolution in the corresponding event-graph.

Whenever a disruptive change occurs, we record the parent *clusterId* (c_p)and child *clusterId* (c)relationship as an 'evolutionEdge' ($c \leftarrow c_p$ is an 'evolutionEdge'). Each 'evolutionEdge' that emerges in the current *time window w,* is processed at the end of the window, and the event summary is updated (cf. Section 6.3). We exploit the graph structure so that only the necessary 'evolutionEdges' are recorded. Note that 'evolutionEdges' track the evolution of event-graphs and they are not the edges in the graph G^t (V^t, E^t). Instead of maintaining complete snapshots, we just maintain 'evolutionEdges' to capture the differences between G^t and G^{t+1}.

We next illustrate how the event summary changes due to addition and deletion of nodes. Similar process is applicable on graph edges.

6.1 Effect of Node Addition

$G^t_c(V^t_c, E^t_c)$ is an event-graph and $G'(V', E')$ is the graph induced on G^t_c such that $E' \subset E^t_c$ be the set of *pivot edges* for graph G^t_c. When a new node (keyword) n joins the event-graph, it may induce a *new* pivot edge in graph G^{t+1}_c. A node n can join the event-graph G^t_c in two possible ways:

Case 1: Due to the addition of node n, an edge $e \in E^{t+1}_c - E'$ becomes a new *pivot edge.*

For example, as shown in Figure 7, when a new node n joins the cluster {A, B, C, D}, a new short cycle $n \rightarrow A \rightarrow B \rightarrow n$ is induced such that edge *AB* becomes a new *pivot edge.*

In this case, the induced graph *G'* includes an extra node (node A) and the corresponding edge(s). The existing *userCover* is extended to cover 'A'. Please note, it is possible that the event summary does not change as the existing *userCover* could be sufficient due to pivot edge *e*(B, C). This check is done in *O* (1).

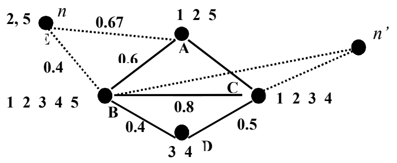


Figure 7: Change in event summary with node addition

Case 2: A node n joins the cluster such that it induces a new cycle on an existing *pivot edges* $e_p \in E'$.

In Figure 7, *n'* induces a new *short cycle* on edge BC ($B \rightarrow n' \rightarrow C$) which was already a pivot edge. Therefore, node *n'* is a peripheral node and no change is made in the event summary.

The cases underline the way event-graph is exploited to update the summary; due to the concept of pivot edges, summary does not change rapidly because of minor changes in the event-graph.

6.2 Effect of Node Deletion

A departing node breaks at least one *short cycle.* Departing node is either a pivot node or a non-pivot node.

Case 1: The departing node n is a non-pivot node (e.g., node n in Figure 7); $n \notin V'$. With the departure of node n, a *pivot edge* may no longer remain the *pivot edge.* Since n is not a pivot node it can impact only one pivot edge as it induces only one cycle. However, the *userCover* continues to remain a valid user cover as the edge $e_{(n', u)}$, erstwhile *pivot edge*, remains part of the event-graph G^{t+1}_c.

Case 2: The departing node n is a *pivot node* for *edge e* (n, v). For example, consider node B in Figure 7. Departure of a *pivot node* has a significant impact on the event-graph G^t_c and the graph may break into multiple sub-graphs or it may get dissolved if it no longer possesses the *short-cycle* property (in that case, the event ceases to exist as a live event). Each surviving sub-graph must possess SCP. Each of the surviving sub-graphs is assigned a new *clusterId.* The *updateEvolutionEdge* () records the relationship between the old and each new *clusterId.* For each 'evolutionEdge' $c_i \leftarrow c$, we extract the event-graph G^{t+1}_{ci}, update its summary and its event thread.

```
Algorithm: UpdatePECover (node n)
$\forall n' \in N(n)$ /*N(n) returns neighbors of n in $G^t_c$*/
  If $n' \in V'$ /* V' is a set of pivot nodes in induced graph G'*/
    $\forall u \in N(n')$
      if $e_{(n', u)} \in E'$
        if $e_p$ is no longer a pivot edge
          $E' \leftarrow E' - e_{(n', u)}$;
      else /*edge is a pivot edge*/
        childId ← getNewClusterId();
        updateEvolutionEdge (childId, clusterId);
```

The merging and splitting of event-graphs, results in an *eventTree* which is a Directed Acyclic Graph (DAG), representing the contextual event summary thread similar to shown in Figure 1.

6.3 Temporal Evolution of Event Thread

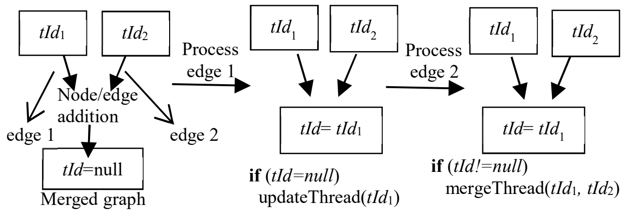


Figure 8: Processing 'evolutionEdges' to update event threads

For all the 'evolutionEdges' (*childId←parentId*), the corresponding event threads are updated. Figure 8 depicts how the event threads evolve with time. Each *eventTree* is identified by a unique id, called *tId*. For each *childId←parentId* 'edge', if the parent cluster's *tId* is null, it is a new event. We initialize its *eventTree* and identify event summary (using approxUserCover). Otherwise, we merge the *eventTree* of the parent cluster with the *tId* of the child cluster and update the event summary. The evolving event summary is recorded in the event thread. Similarly, event threads are updated in case of event-graph split. For two *clusterId*s c_1 and c_2 in an *eventTree,* if c_1 is ancestor of c_2, c_1 has occurred before c_2 in real world time. Each path in an event thread, from each of its roots to each of its leaves, exposes a different facet of the event. The event thread in Figure 1 has 6 facets.

7. PERFORMANCE EVALUATION

The goals of our experiments are to study our system's ability a) to construct informative, complete, meaningful, stable and compact event summaries in real time (Section 7.2); b) to discover the contextual event summary threads in real time (similar to Figure 1) and to study the impact of the changes in the granularity of the event-graph on the event summary and event threads (Section 7.3); c) to discover event threads efficiently and in a scalable manner (Section 7.4). The experiments use the prototype built by us [31] and run on a quad-core 2.61 GHz, 4GB RAM machine running Windows 8 and Java as programming language.

In Table 2, we describe the Twitter traces used in experiments. We use two types of traces: a) general timeline based (ALL) which contains all the tweets generated within US geography within a time window, provided by Twitter API and b) event specific (ES) traces. The event specific traces were created as follows: For each event, we specify a set of rules. Each rule contains one or more relevant keywords and/or hashtags associated with the event. Any tweet, containing the keywords from a rule is included in the event trace. We have carefully selected the traces to cover the entire spectrum of event change density -- from very low (22) to very high (775). **Events change density** specifies total number of times *all* the underlying event-graphs in a trace evolve every 100k tweets. Traces are read in their chronological order to mimic the real-time arrival of the tweets.

Table 2: Details of Datasets

Event	*#of Tweets*	*Events change density*
Big Data	200k	775 changes/100k tweets
Nairobi–complete	720k	274 changes/100k tweets
Syria	1.7million	182 changes/100k tweets
Twitter Time Line (ALL)	3.2million	22 changes/100k tweets

7.1 Discovering Base Events

To the best of our knowledge, no previous system exists that summarizes a complete live data stream in the absence of any user query. Therefore, we construct the ground truth as follows: Live events unraveling in the data stream are the base events and form the ground truth for our system. The objective of our experiments is to study the performance of our summarization technique and its ability to construct meaningful *contextual summary threads* for the events given to it as ground truth. Any algorithm discovering dense graphs as event-topics in a highly dynamic graph can be used to provide the base events. For our experiments, we use the algorithm in [15] – it efficiently discovers the events in a live data stream with high precision and recall. It is shown in [15] that the events discovered by this system correlates highly with real world events reported in Google news headlines. Additionally, it discovers many other real world events that do not occur in Google news headlines.

The number of event-topics as well as the number of keywords in an event-topic in a data stream depends on the dynamic event-graph $G^t(V^t, E^t)$ at time *t*. The set of nodes V^t in the graph G^t depends on burstiness threshold γ and the set of edges E^t depends on the edge similarity threshold λ. The default values are: γ=5 and λ=0.2, unless specified otherwise. The message block size *m* is set to 1000 and *w* (cf. Table 1) is set to be 75 for all the experiments.

7.2 Quality of Event Summarization

Informative-ness: We identify the *userCover* only for the pivot nodes in an event-graph (Section 5). The premise is that the important keywords in the event-graph are likely to be pivot nodes. The peripheral nodes in the event-graph may or may not be covered in the event summary. We quantify the informative-ness of the summary as follows: If there is a keyword in an event-topic that is a proper noun and is not present in the event summary, we count it towards loss of precision. Precision is computed as the fraction of proper noun keywords present in the event summary among all the proper noun keywords in the event-topic. Let there be N events in a given trace. Let R_i be the set of proper noun words in the summary of the i^{th} event discovered by our algorithm; $1 \le i \le N$. Let B_i be the set of all the proper noun keywords present in the event-topic of event *i*. The precision of informative-ness, P_I, is defined as:

$$P_I = \frac{\sum |R_i|}{\sum |B_i|}; 1 \le i \le N$$

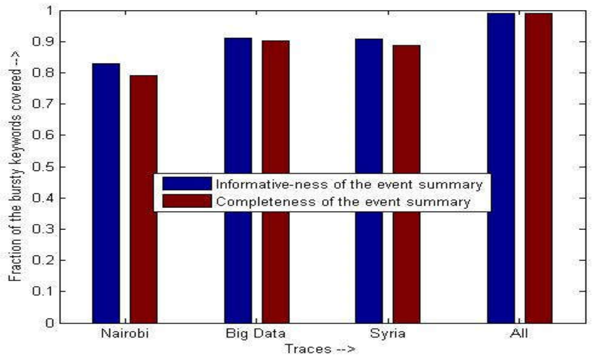


Figure 9: Summary Informative-ness and Completeness

Complete-ness: We compute the fraction of *total* keywords in the event-topic covered in the event summary to compute the completeness score P_C. As shown in Figure 9, P_C for various traces varies from 79% to 99%. P_C is marginally lower than P_I, as typically noun keywords are more central in the event-graphs. For the ALL trace, since almost all the keywords are covered in the event summary, there is no difference in the two scores. We see, *that our system achieves very high informative-ness and complete-ness score.*

Stability: We study the stability of the event summaries by our algorithm for live events. The results are shown in Table 3. Since different traces have different sizes, we report the results in terms of event change density/100k tweets.

Table 3: Rate of changes in event-graphs for different traces

Total event-graph changes/ 100k tweets	Changes (addition in event-graph)	Changes (event-graph break)	No change in summary (additions)	No change in summary (deletions)
Big data-775	114	33	411	217
Nairobi-274	43	5	173	53
Syria-182	30	3	114	35
ALL- 22	1.5	0	13.5	7

We see that across the traces, more than 80% of the changes in the event-graphs do not result in any change in the summary. For example, for Big-Data trace, for every 775 changes in the event-graphs; event summary remains the same for 411 changes when nodes/edges or messages get added to the event-graph and for 217 changes when a node/edge gets deleted from the event-graph, i.e., no summary change for 81% of event-graph changes. Thus, *the event summary remains stable for a large fraction of changes.*

Summary-size: Next, we compare the message pool size of an event with the number of messages in its summary (summary-size). The average message pool size varies from 74 tweets (for ALL trace) to 1394 tweets (for Syria trace) as shown in Figure 11, denoting that for an emerging event in the Twitter data stream, a large number of related messages are posted. The number of messages is per event-graph and not per event thread (an event thread captures the evolution of associated event-graph(s)). We divide the events based on the number of keywords in the event-topic, as shown in Figure 10 and Figure 11. For ALL trace, no event-topic contained more than 16 keywords.

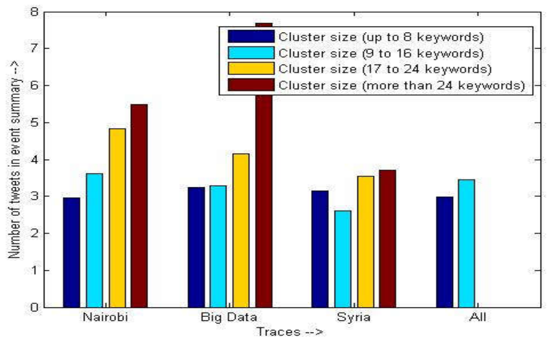


Figure 10: Average number of tweets in 'event summary' for different event-topics sizes

As expected, summary-size increases with increasing event-topic size (Figure 10). Similarly, number of messages pertaining to an event increases with event cluster size (Figure 11). The key insights are:

1) Summary-size is independent of the message pool size (i.e., tweets associated with an event-graph). It depends only on the underlying information. For example, for Syria trace, average message pool size increase from 128 to 1394 for different size event-topics but the summary-size remains almost stable. The event summaries were highly meaningful.

2) Average summary-size varies from 2.61 to 7.71 tweets for different traces for event-topics comprising up to a few hundred tweets. Thus, our system discovers highly compact summaries.

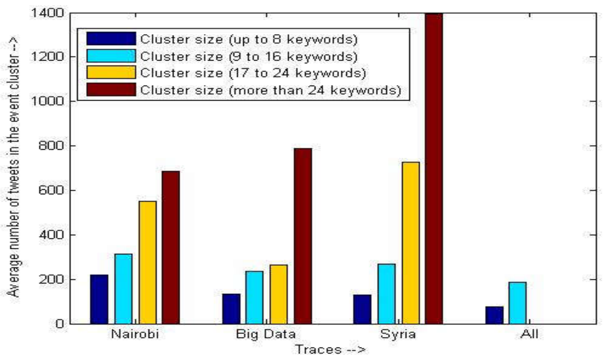


Figure 11: Average number of tweets in the event 'message pool' for different sizes of event-topics

We construct alternative summary for a discovered as follows: For an event-graph, we randomly select a message from the message pool covering an event keyword. We keep on selecting messages till each keyword in the event graph is covered by at least one message. This 'naïve method' serves as a baseline to compare the performance of our system.

Since we select the messages randomly in 'naïve method', qualitatively, the summary by naïve method was markedly inferior for almost all the events. The other issues with this naïve method were; 1) A significant number of redundant messages occur in the event summary. For many events, the summary size was more than twice the summary discovered by our system.; 2) the naïve method will not discover the event threads, exposing different event facets.

To further compare the quality of event summary we compared the summary discovered based on our approach with Google News headlines (for the events which also appear in Google headlines). In Table 4, we present the comparisons between a few of the Google headlines with our summary tweets.

Table 4: Google News Headlines Vs. Event Summary discovered by our approach

Google Headline	Summary based on our approach
India vs New Zealand, 3rd Test: Ashwin 6/81 hands India huge first innings lead	India vs New Zealand, 3rd Test: Ashwin 6/81 hands India huge first innings lead https://t.co/ih5oTkOlIY
NASA Mission Tests Thrusters On Journey To Asteroid	NASA probe tests thrusters on journey to asteroid Bennu - Zee News: NASA probe tests thrusters on jou... https://t.co/2Tv0XC71cJ
Pampore attack: Militants holed up inside govt building; combing operations intensify	RT @kashmirglobal: Smoke and dust engulf a government building where suspected militants have fighting with Indian forced in Pampore…
NASA resupply mission to space station postponed	Atlantic Storm System Delays NASA Resupply Launch to Space Station via NASA https://t.co/wdWmqwhz7k
Obama pushes NASA to send humans to Mars by 2030s	Can the U.S. Really Get Astronauts to Mars by 2030?: President Obama renewed his call to send Americans to th... https://t.co/ZeSADfvDNZ
1000 asteroids heading towards Earth; conspiracy theorists claim end of the world is near!	RT @Ufo_area: Asteroid mission: 1000 space rocks heading towards Earth – Daily Star https://t.co/ExDVSJOzly #Asteroid

Microsoft Office for Android will be supported on Chrome OS	Microsoft Office for Android will be supported on Chrome OS +AC0- The Indian Express https://t.co/7xLBJJpSKW

We see that the summary tweets represent the Google headlines very accurately. For a few headlines, Google headline matches completely with the tweet selected by our approach. However, the timestamp of tweets in our summary is ahead by few minutes to few hours, for different event, compared to their corresponding appearance in Google headlines. Details are omitted due to lack of space.

In our next experiment, we study the performance of our system to expose context event summary threads.

7.3 Contextual Event Summary Threads

How do Real-time Events Evolve with Time? – Next, we study what fraction of the real-time events evolve into event threads. We divide the events into; 1) standalone events, i.e., events which do not result in threads and 2) event threads. We plot the density of events per 100k tweets for each trace. The results are shown in Table 5. The event density is highest for Big-data trace. We see; 1) a significant number of events result in event threads; and 2) the density of events is much higher for event specific traces as opposed to ALL trace. ALL trace has a density of 1.1 standalone events and 1 event thread/100k tweets respectively. Since the average tweets/event is highest for Syria trace (Figure 11), the density of events for it is relatively smaller.

We show the average number of times an event-graph changes during its life cycle in Table 5. An interesting insight is that even though the density of events is lowest for ALL trace, the average number of times its event-graphs changes during their life span is significantly higher compared to ES traces. The reason is, density of tweets related to a single event in ALL trace is very low. Therefore, the changes in the existing event-graphs are only marginal and the event-graphs absorb such changes, resulting in a longer life span. In summary, *a large fraction of events result in threads. Further event threads evolve only when there are significant changes in the event.*

Table 5: Event density and Average number of times an event-topic changes during its life span

Density/100K Tweets	Nairobi	Big Data	Syria	All
Density of Standalone Events	19.44	94.38	16.25	1.1
Density of Event Threads	39.9	59.9	15.6	1
# of Changes in an event-topic during its life span	6.22	5.88	6.46	10.88

How Complete are the Event Threads? - In this experiment, we study how the real-time events evolve. An event thread is a DAG. The depth of a DAG is the length of the path from its root to its deepest leaf (depth of DAG in Figure 1 is 9). To count the number of facets in the DAG, we sum the total number of unique paths from the root(s) to each of the leaf nodes of an event thread. We study the temporal evolution of the events by computing the average depth and average number of facets of the event threads. The facets are counted as: $facets = \sum_{l \in L} \sum_{r \in R} p_l^r$ where p_l^r is total number of paths to reach from root node r to leaf node l and L (R) is the set of leaf (root) nodes in the event-graph. Depth of an event thread and its facets capture the complexity of the events. The results for different datasets are shown in Table 6. Only the event threads, not the stand-alone events, are considered for this experiment.

Table 6: Average depth and average facets for an event

Event Thread Complexity	Nairobi	Big Data	Syria	All
Average event depth	5.98	4.97	5.42	2.77
Average event facets	2.63	2.25	2.27	1.21

Average depth/facets of the event threads are highest for Nairobi trace at 5.98/2.63 and lowest for ALL trace at 2.77/1.21. Hence, event complexity is highest for Nairobi trace. The result signifies that our algorithm can handle changes in a fast- moving data stream gracefully. For the default values of λ and γ, the maximum depth of an event was 30 (for Nairobi trace) with 9 facets, exposing the complex way the live events evolve.

How does Granularity of Event-Graph impact the summary? To vary the granularity of the dynamic graph *G^t (V^t, E^t),* we vary the burstiness threshold (Bt) γ and edge correlation threshold (Ec) λ. If γ and/or λ are reduced, there will be more nodes and/or edges in the graph *G^t (V^t, E^t),* leading to more events being discovered and vice versa. In Figure 12, we show how the number of event threads (every 100k tweets) varies -- by varying γ and λ for different traces.

We find two distinct trends: 1) for Big-Data and Syria traces, with more nodes/edges in the dynamic graph G^t (lower γ and/or λ), the number of event threads increase but the depth and the facets are not significantly impacted; 2) for Nairobi and ALL traces, with more nodes/edges in G^t, the number of events is not impacted much but the event depth/facets increase, exposing the same events at finer granularity. In summary, *when the messages related to an event tend to come in bursts, we observe trend* (1). *If the messages related to an event are distributed more evenly in the data stream, we observe trend* (2).

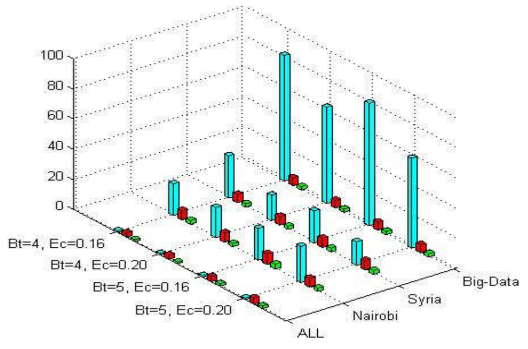


Figure 12. Event Density (blue), event thread depth (red) and event thread facets (green) with varying γ (Bt), λ (Ec)

7.4 Efficiency and Scalability

In this experiment, we analyze the overhead of our method to summarize the events and arrange them into contextual summary threads. We also analyze the scalability of our approach, i.e., how many tweets are processed/second (TPS). We discover the event-topics in a twitter trace without contextual summarization system and with it. In Table 7, we show the overhead of our system. We see that the summarization algorithm imposes only a marginal overhead over base event discovery system in [15]. Overhead of our method is highest for Nairobi trace at 12.36% and smallest for ALL trace at 3.17%. The tweet processing rate is highest for ALL trace at 6631 TPS and lowest for Big-Data trace at 1044 TPS, with summarization system on. For Nairobi trace, the rate is 4473 TPS with summarization and 5026 without summarization.

Table 7: Tweet processing rate per second (TPS)

TPS	Nairobi	Big Data	Syria	All
With Summarization	4473	1044	2505	6631
Without Summarization	5026	1129	2614	6841

Without the summarization, the TPS for ALL trace is 6841 for the algorithm presented in [15]. ALL trace is closest to the general Twitter data stream. The average rate for global Twitter data stream is reported to be 5700 TPS in August 2013 (peak rate is ~144k TPS) [26]. Therefore, *our technique is highly scalable for real time processing and does not impose a big overhead to identify event summary and event threads* over event discovery algorithm.

In summary, we present a novel system that constructs meaningful, stable, and compact event summaries for the events present in an unfiltered data stream. We also discover contextual event threads in real time over live data streams efficiently.

8. CONCLUSION

In this paper, we present a novel unsupervised technique that builds the summaries for emerging events in real time in a complete fast moving data streams in absence of any user query. The summaries are complete and meaningful and contain the informative messages for the underlying events. Our technique also discovers the contextual event summary threads in a scalable manner. It is not necessary that the most recent messages are also the most informative for a live event. However, the most informative messages about the event are present in its summaries. We plan to extend our technique to enable improved real time search over data streams.

9. ACKNOWLEDGMENTS

Pranjal Khare (student, Department of CSE, IIT-Bombay) used our system to produce results presented in Table 4.

10. REFERENCES

[1] Ramesh Nallapati, Ao Feng, Fuchun Peng, and James Allan, "Event threading within news topics", in CIKM, 2004.

[2] Ao Feng, and James Allan, "*Finding and linking incidents in news"*, CIKM 2007, page 821-830.

[3] W. Gao, P. Li, K. Darwish, "Joint Topic Modeling for Event Summarization across News and Social Media Streams", in CIKM 2013.

[4] Chen Lin et al., "Generating Event Stroylines from Microblogs", in CIKM 2012, page 175-184.

[5] L. Shou, Z. Wang, K. Chen, G. Chen, "Sumblr: Continuous Summarization of Evolving Tweet Streams", in SIGIR 2013.

[6] Yajuan Duan et al., "An Empirical Study on Learning to Rank of Tweets", in COLING 2010.

[7] J. Vosecky, D. Jiang, K. W. Leung, W. Ng, "Dynamic Multi-Faceted Topic Discovery in Twitter", in CIKM 2013.

[8] X. Yang, A. Ghoting, Y. Ruan, S. Parthsarthy, "A Framework for Summarizing and Analyzing Twitter Feed", in SIGKDD 2012.

[9] K. Kutzkov, R. Pagh, "Triangle counting in dynamic graph streams", in SWAT 2014.

[10] B. Sharifi, M. Hutton, J. Kalita,. "Summarizing Microblogs Automatically", in NAACL-HLT, 2010.

[11] D. Chakrabarti, K. Punera, "Event Summarization using Tweets", in ICSWM 2011.

[12] Z. Yang, K. Cai, J. Tang, L. Zhang, Z. Su, J. Li, "Social Context Summarization", in SIGIR 2011, pages 255-264.

[13] F. T. Chua, S. Asur, "Automatic Summarization of Events from Social Media", ICWSM 2013.

[14] M. Mathioudakis, N. Koudas, "TwitterMonitor: Trend Detection over the Twitter Stream", SIGMOD 2010.

[15] M. K Agarwal, K. Ramamritham, M. Bhide "Real Time Discovery of Dense Clusters in Highly Dynamic Graphs: Identifying Real World Events in Highly Dynamic Environments", in VLDB 2012.

[16] Bansal N., Chiang F., Koudas N., Tompa F. "Seeking Stable Clusters in the Blogosphere", in VLDB 2007.

[17] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment", in Journal of the ACM (JACM), Vol. 46, 1999.

[18] H. Kwak, C. Lee, H. Park, S. Moon, "What is Twitter, a Social Network or a News Media?", in WWW 2010.

[19] C. Chen, F. Li, B. C. Ooi, S. Wu, "T1: An Efficient Indexing Mechanism for Real-Time Search on Tweets", in SIGMOD 2011.

[20] X. Hu, Y. Tao, C.-W. Chung, "Massive Graph Triangulation", in SIGMOD 2013.

[21] S. Brin, L. Page, "The Anatomy of a Large-Scale Hyper textual Web Search Engine", J. of Computer Networks, Vol. 30, 1998.

[22] U. Brandes, "A Faster Algorithm for Betweenness Centrality", J. of Mathematical Sociology, 25(2), 163-177, 2001.

[23] J. Xu, D. Kalashnikov, S. Mehrotra, "Efficient Summarization Framework for Multi-Attribute Uncertain Data", in SIGMOD 2014.

[24] P. Lee, L. Lakshmanan, E. Milios, "Incremental Cluster Evaluation Tracking from Highly Dynamic Network Data", in ICDE 2014.

[25] http://www.nada.kth.se/~viggo/wwwcompendium/

[26] https://blog.twitter.com/2013/new-tweets-per-second-record-and-how

[27] A. Pavan, et al., "Counting and Sampling Triangles from a Graph Stream", in VLDB 2014.

[28] N. Wang et al., "On Triangulation-based Dense Neighborhood Graphs Discovery", in PVLDB, 4(2):58–68, 2010.

[29] http://en.wikipedia.org/wiki/Chordal_graph

[30] https://en.wikipedia.org/wiki/Westgate_shopping_mall_attac k

[31] M. K. Agarwal, D. Bansal, M. Garg, K. Ramamritham, "Keyword Search on Microblog Data Streams: Finding contextual Messages in Real Time", in EDBT 2016.