CAST: A Context-Aware Story-Teller for Streaming Social Content

Pei Lee  
University of British Columbia  
Vancouver, BC, Canada  
peil@cs.ubc.ca

Laks V.S. Lakshmanan  
University of British Columbia  
Vancouver, BC, Canada  
laks@cs.ubc.ca

Evangelos Milios  
Dalhousie University  
Halifax, NS, Canada  
eem@cs.dal.ca

ABSTRACT

Online social streams such as Twitter timelines, forum discussions and email threads have emerged as important channels for information propagation. Mining transient stories and their correlations implicit in social streams is a challenging task, since these streams are noisy and surge quickly. In this paper, we propose CAST, which is a context-aware story-teller that discovers new stories from social streams and tracks their structural context on the fly to build a vein of stories. More precisely, we model the social stream as a capillary network, and define stories by a new cohesive subgraph type called \((k, d)\)-Core in the capillary network. We propose deterministic and randomized context search to support the iceberg query, which builds the story vein as social streams flow. We perform detailed experimental study on real Twitter streams and the results demonstrate the creativity and value of our approach.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—Data Mining;  
H.3.3 [Information Storage and Retrieval]: Information and Retrieval—Search process

General Terms

Measurement, Experimentation

Keywords

Story Vein; Context-aware; Story-teller; Social content

1. INTRODUCTION

In the current social web age, the fast surge of online streaming social content, such as Twitter/Facebook timelines, blog posts, forum discussions and email threads, may easily lead to “information anxiety”, which is the gap between the information we receive and the information we are able to digest [11]. To relieve the information anxiety, there is an emergent need to organize streaming social content in a structural way and mine the “real gold” out of noisy buzzes. There are many previous studies [14, 18, 22, 28] focusing on detecting new emerging events from social streams. They serve the need for answering “what’s happening now?”. However, in reality, events usually do not happen in isolation, and existing studies fail to track the correlation between them. For example, “Crimea votes to join Russia” (on May 6, 2014) and “President Yanukovych signs compromise” (on February 21, 2014) are two separate events, but they are actually highly correlated under the same topic “Ukraine Crisis”. In this paper, our goal is to design a context-aware story-teller for streaming social content, which not merely detects trending stories in a given time window of observation, but also builds the “context” of each story by measuring its correlation with other stories on the fly. As a result, our story-teller has advantages on responding advanced user queries like “tell me related stories”, which is crucial to help digest social streams.

Building a context-aware story-teller over streaming social content is a highly challenging problem, as explained below:

- **The effective organization of social content.** It is well known that posts such as tweets are usually short, written informally with lots of grammatical errors, and even worse, a correctly written post may have no significance and be just noise.
- **Identification of transient stories from time window.** Story detection should be robust to noise and efficient to support single-pass tracking essential in the streaming environment.
- **Story context search on the fly.** Story correlation computation should be interpretable and efficient to support online queries.

To the best of our knowledge, no training data set for the context-aware story-telling on streaming social content is publicly available, which renders the existing works [20, 24] on Story Link Detection (SLD) not applicable, because SLD is trained on well-written news articles. Furthermore, we cannot apply topic tracking techniques [10, 8] to story context search, because topic tracking is usually formulated as a classification problem [2], with an assumption that topics are predefined before tracking, which is unrealistic for story context search on social streams.

To address above challenges, we propose CAST in this paper, which is a Context-Aware Story-Teller specifically designed for streaming social content. CAST takes a noisy social stream as the input, and outputs a “story vein”, which is a human digestible and evolving summarization graph by linking highly correlated stories. The major workflow of CAST is illustrated in Figure 1. First of all, we treat each post as a node and add a “capillary” edge between two posts if they are similar enough. For example, “Australian authorities update search for MH370” and “new MH370 search area is closer to Australia” are two similar posts and we add a capillary edge between them. By this way, a capillary network will be constructed for social posts in the same observing time window. Second, we propose a new notion of \((k, d)\)-Core to define a transient story, which is a cohesive subgraph in a capillary network. In
2. MODELING SOCIAL STREAMS

Social streams such as those found in microblogging sites like Twitter and discussion forums are typically noisy, are of large volume, and surge quickly. In this section, we describe how we model social streams and formalize the main problem studied in this paper. We use the term post to denote each object in a social stream.

2.1 Social Capillary

We begin by formalizing the social stream, social capillary and network as follows.

DEFINITION 1. (Social Stream) A social stream $Q$ is a first-in-first-out queue of posts ordered by time of arrival, in which each post $p$ is represented as a triple $(p^t, p^r, p^n)$, where $p^t$ is the text content of $p$, $p^r$ is the time stamp, and $p^n$ is the author of the post.

The next definition assumes that we have a measure $s(p_i, p_j)$ to gauge the similarity between a pair of posts $p_i$ and $p_j$. Examples of post similarity are given below.

DEFINITION 2. (Social Capillary and Network) Given two posts $p_i, p_j$ in a social stream $Q$ and a threshold $\epsilon$ ($0 < \epsilon < 1$), there is a social capillary between $p_i$ and $p_j$ if the post similarity $s(p_i, p_j) \geq \epsilon$. The capillary network corresponding to $Q$ is denoted as $G(V, E)$, where each node $p \in V$ is a post in $Q$, and each edge $(p_i, p_j) \in E$ is a social capillary with weight $s(p_i, p_j)$.

Intuitively, a social capillary connects two posts if they are similar enough. The capillary network can be viewed as a structural representation of the original unstructured social stream, by organizing meaningful information from noisy buzzes. Specifically, posts with few capillaries can be essentially treated as noise and ignored. There are several key problems to tackle towards the effective modeling of social streams, as discussed below.

Entity Extraction. Our goal is to design a processing strategy that can quickly judge what a post talks about and is robust enough to the informal writing style. In particular, we focus on entities in a post, since entities depict the story. However, traditional Named Entity Recognition tools [19] only support a narrow range of entities like Locations, Persons and Organizations. Also, simply treating each post text as a bag of words [17] will lead to loss of accuracy, since different words have different weights in describing a story. To broaden the applicability, we treat each noun in the post text as a candidate entity. For example, given a tweet “iPad 3 battery pointing to thinner, lighter tablet?”, the entities are “iPad”, “battery” and “tablet”. Technically, we obtain an entity list $p^T$ from a post $p$ using a Part-Of-Speech Tagger$^1$.

Post Similarity. Traditional similarity measures such as TF-IDF based cosine similarity, Jaccard Coefficient and Pearson Correlation [17] only consider the post content. However, time stamps should play an important role in determining post similarity, since posts created closer together in time are more likely to discuss the same story than posts created at very different moments. We define the similarity between a pair of posts $p_i$ and $p_j$ by combining both content similarity and temporal proximity, that is

$$s(p_i, p_j) = s_T(p_i^T, p_j^T) \cdot s_s(|p_i^T - p_j^T|)$$

where $s_T(p_i^T, p_j^T)$ is a set-based similarity measure (e.g. Jaccard Coefficient [17]), and $s_s(|p_i^T - p_j^T|)$ is a distance measure monotonically decreasing with $|p_i^T - p_j^T|/\Delta t$, where $\Delta t$ is the time window sliding step size, typically in hours. Both $s_T(p_i^T, p_j^T)$ and $s_s(|p_i^T - p_j^T|)$ fall into the interval $[0, 1]$. In this paper, we select $s_T(p_i^T, p_j^T) = (|p_i^T \cap p_j^T|)/(|p_i^T \cup p_j^T|)$ and $s_s(|p_i^T - p_j^T|) = e^{-|p_i^T - p_j^T|}$. It is trivial to see $0 \leq s(p_i, p_j) \leq 1$.

Capillary Network Construction. In this paper, we assume that the social stream $Q$ is monitored by a sliding time window of observation. Since new posts flow in and old posts fade out at each moment, the capillary network $G(V, E)$ in the time window is dynamically updated, with new nodes/edges added and old nodes/edges removed. Removing a node and associated edges from $G(V, E)$ is an easy operation. Let us investigate the case when a node is added.

\footnote{POS Tagger, http://nlp.stanford.edu/software/tagger.shtml}
When a new post \( p \) flows in, we need to construct the capillaries between post and nodes in \( V \), following Definition 2. In a real scenario, since the number of nodes \( |V| \) can easily go up to millions, it is impractical to compare \( p \) with every node in \( V \) to verify the satisfaction of capillary. To solve this problem, we build inverted index on (entity, post) pairs and only check similarities between \( p \) and those posts sharing common entities with \( p \).

### 2.2 Story Vein

In this paper, we propose CAST, an effective context-aware story-teller for social streams. As social posts flow in, CAST is able to discover new stories and track the "vein" between stories, in which each story is a group of highly similar posts telling the same thing in the social stream, and each vein is a correlation link between two stories. We define a story vein as follows, where we use \( \text{Cor}(S_i, S_j) \) to denote the correlation between stories \( S_i \) and \( S_j \).

**Definition 3. (Story Vein)** Given a social stream \( Q \) and a threshold \( \gamma (0 < \gamma < 1) \), the output of CAST can be represented by a directed graph \( G(V, E) \), where each node \( S \in V \) is a story, and each edge \( (S_i, S_j) \in E \) means the story correlation \( \text{Cor}(S_i, S_j) \geq \gamma \) and \( S_i \) happens earlier than \( S_j \).

The motivation behind CAST is, real world stories are not isolated but commonly correlated with each other. Intuitively, the context of \( S \) in \( G(V, E) \) is the neighboring stories of \( S \). In particular, we use upper and lower context to represent the stories connected by incoming and outgoing edges of \( S \), respectively. The target of CAST is to discover new stories and build the contexts of these stories. As the social stream \( Q \) gets updated over time, the story vein \( G(V, E) \) will also be updated in real time. The dynamic nature of context-aware story-telling raises challenges both in the discovery of new stories and in the tracking of story context. In the following, we will address these challenges by discussing transient story discovery in Section 3 and story context tracking in Section 4.

### 3. TRANSIENT STORY DISCOVERY

In this section, we describe the motivation and algorithms for identifying transient stories as a new kind of cohesive subgraph, \((k,d)-\text{Core}\), from the capillary network.

#### 3.1 Defining a Story

Edge weights in a capillary network have very natural semantics: the higher the post similarity, the more likely two posts are to talk about the same story. It is well-known that a cohesive subgraph is a sub-structure with high edge density and very low internal commute distance [27]. Suppose \( S(V_S, E_S) \) is a cohesive subgraph in \( G(V, E) \). Since nodes in \( S(V_S, E_S) \) have a high density to connect with each other, it is very likely that all nodes in \( S(V_S, E_S) \) share the same content and tell the same story. Based on this observation, we model a story in social stream as a cohesive subgraph in capillary network in this paper.

There are many alternative ways to define a cohesive subgraph. **Clique** may be the best candidate in spirit, because any two nodes have a sufficiently high similarity and thus share the same content. However, in real world data sets, cliques are too restrictive for story definition and this calls for some relaxations. There are several choices for clique relaxations, e.g., quasi-clique, \( k \)-Core, \( k \)-Plex, etc [13]. Given a connected subgraph \( S(V_S, E_S) \), supposing \( N(p) \) is the neighbor set of node \( p \in V_S \) in \( S(V_S, E_S) \), these clique relaxations are defined as:

- **Quasi-clique**: \( |N(p)| \geq \lambda |V_S| - 1 \) for every post \( p \in V_S \), where \( 0 < \lambda \leq 1 \);
- **k-Core**: \( |N(p)| \geq k \) for every post \( p \in V_S \);
- **k-Plex**: \( |N(p)| \geq k \) for every post \( p \in V_S \).

Notice that finding the maximum clique, quasi-clique or \( k \)-Plex from a graph is an NP-Hard problem [7, 4]. Even worse, [9] proved that there are no polynomial algorithms that provide any reasonable approximation to the maximum clique problem. Since a clique is a special case of quasi-clique or \( k \)-Plex, these hardness results carry over to maximum quasi-clique and maximum \( k \)-Plex problems. There are a lot of heuristic algorithms that provide no theoretical guarantee on the quality [7, 1, 4]. Since most of them are based on local search [7], they do not scale to large networks, because local search involves the optimization of solutions by iteratively moving to a better neighbor solution in an exponential search space.

The good news is that \( k \)-Cores can be exactly found in polynomial time. By adjusting \( k \), we can generate \( k \)-Cores with desired edge density \( 2|E_S|/|V_S| \). For example, increasing \( k \) will improve edge density because the minimal edge degree is increased and \( |V_S| \) is decreased in the same time. However, \( k \)-Cores are not always capable of capturing our intuition on stories: although each post has at least \( k \) neighbors, the fact that two posts are connected by a single capillary may be not a strong enough evidence to prove that they tell the same story. Sometimes, posts only share some common words but discuss different stories, e.g., “Google acquires Motorola Mobility” and “Bell Mobility acquires Virgin Mobile”.

We show an example of 3-Core in Figure 2(a), where \( p_1 \) and \( p_2 \) are connected but they belong to two separate cliques. To address this challenge, we make a key observation that the existence of more common neighbors between two capillary-connected posts suggest a stronger commonality in story-telling. Supposing \( p_1 \) and \( p_2 \) are connected by a capillary and there exists post \( p_t \in N(p_1) \cap N(p_2) \), then we call \( p_t \) a witness for the post similarity \( s(p_1, p_2) \). We capture this intuition in the following definition, where we formalize a story as a \((k,d)\)-Core.

**Definition 4. (Story)** A story in social stream \( Q \) is defined by a maximal \((k,d)\)-Core \( S(V_S, E_S) \) in capillary network \( G(V, E) \) associated with \( Q \), where \( k, d \) are numbers with \( k > d > 0 \) and

- \( S(V_S, E_S) \) is a connected subgraph
- For every post \( p \in V_S \), \( |N(p)| \geq k \)
- For every edge \( (p_i, p_j) \in E_S \), \( |N(p_i) \cap N(p_j)| \geq d \)

**Why \((k,d)\)-Core?** In \((k,d)\)-Cores, we use \( k \) to adjust the edge density, and use \( d \) to control the strength of similarity witness. It is easy to see that a maximal \((k,d)\)-Core is a subgraph of a maximal \( k \)-Core. However, compared with \( k \)-Cores, \((k,d)\)-Cores have a more cohesive internal structure enhanced by at least \( d \) common neighbors as witnesses of commonality between two nodes connected by an edge. This enhancement makes posts in \((k,d)\)-Core more likely to tell the same story. We show an example of 3-Core and (3,1)-Core in Figure 2(b). As we can see, \( p_3 \) is in 3-Core but not in (3,1)-Core, because the similarity between \( p_1 \) and \( p_4 \) is not witnessed by other posts. Besides, [21] defines a kind of cohesive subgraph called k-Dense, in which every edge has at least \( k \) witnesses. It is easy to infer that a k-Dense is a \((k+1,k)\)-Core. Thus, k-Dense is a special case of \((k,d)\)-Core, but provides no flexibility to adjust \( k \) and \( d \) independently. Therefore, our proposed \((k,d)\)-Core is better than k-Core and k-Dense in capturing a story.

**Non-overlap Property.** Similar to maximal \( k \)-Cores, a maximal \((k,d)\)-Core cannot be a subgraph of any other \((k,d)\)-Cores. Maximal \((k,d)\)-Cores have the following important property.

**Proposition 1.** The maximal \((k,d)\)-cores of a graph are pairwise disjoint.
3.2 Story Formation

We now discuss the computation of \((k, d)\)-Cores. We first review the \(k\)-Core generation process \cite{13}: given a network \(G(V, E)\), iteratively remove the nodes with degree less than \(k\) from \(G(V, E)\), until all the remaining nodes have a degree at least \(k\). The result will be a set of maximal \((k, d)\)-Cores, which are obtained in polynomial time. The \(k\)-Core generation process forms the basis of our \((k, d)\)-Core generation algorithms. We propose the first solution for \((k, d)\)-Core generation in Algorithm 1. We call it a Zigzag algorithm, because the basic idea is to repeatedly change the property of the current network between two states: the first state is \(k\)-Core set \(G_1\) obtained by removing nodes recursively, and the second state is \((d + 1, 1)\)-Core set \(G_2\) obtained by removing edges recursively. This Zigzag process will terminate if each connected component in the result set is a \((k, d)\)-Core, or the result set is empty. It is easy to see Algorithm 1 takes polynomial time and the result is exact.

We notice that the transition costs between two states are not symmetric: the computational costs of \(G_1\) and \(G_2\) are very different. If we can reduce the frequency of the transition with a higher overhead, the overall performance of Zigzag can be optimized. The following lemma formalizes the property.

**Proposition 2.** Given a network \(G(V, E)\) with \(|V| < |E|\) and integers \(k, d\) \((k > d)\), for each iteration in Zigzag, the computation of \(k\)-Core set \(G_1\) is more efficient than the computation of \((d + 1, 1)\)-Core set \(G_2\).

**Proof.** To compute \(G_1\), we need to check the degree of every node. To compute \(G_2\), we need to check the common neighbor size of the two end nodes for every edge. Given that \(|V| < |E|\) in most networks and the checking of node degree is more efficient than the checking of common neighbors, we complete the proof.

Proposition 2 suggests that if we reduce the computation frequency of the \((d + 1, 1)\)-Core set \(G_2\), the performance will be improved. Following this, we propose Algorithm 2 to improve the performance of Algorithm 1, by applying node deletions as much as possible. The heuristic is, whenever an edge is deleted, we check whether this deletion makes the degree of its end nodes smaller than \(k\), and if it does, a recursive node deletion process starting from end nodes will be performed (Line 7). We call Algorithm 2 NodeFirst because it greedily invokes the node deletion process when possible. Since NodeFirst avoids to perform a complete edge deletion process as Zigzag did, the network converges very fast to the set of maximal \((k, d)\)-Cores. The following propositions indicate that NodeFirst produces exactly the same results as Zigzag, but its performance is better than Zigzag.

**Proposition 3.** Given a network \(G(V, E)\) and numbers \(k, d\) \((k > d)\), both Zigzag and NodeFirst generate the same result, which is the set of all maximal \((k, d)\)-Cores in \(G(V, E)\).

**Proof.** Suppose \(S_{(k,d)}\) is the set of all maximal \((k, d)\)-Cores in \(G(V, E)\). Both Zigzag and NodeFirst try to remove nodes or edges in each iteration, so the sizes of nodes and edges follow a monotone decreasing trend. With this message, we show the proof by following algorithm logics: Zigzag converges to \(S_{(k,d)}\) because \(S_{(k,d)}\) is the first time when a \(k\)-Core set and a \((d + 1, 1)\)-Core set are equal (Line 7 in Algorithm 1); NodeFirst converges to \(S_{(k,d)}\) because it is the first time when no edges with witness less than \(d\) can be found in a \(k\)-Core set (Line 8 in Algorithm 2).

**Proposition 4.** On each iteration, NodeFirst is more efficient than Zigzag.

**Proof.** We measure the efficiency by the number of node degree check or edge similarity witness check operations. Zigzag takes \(|V| + |E|\) operations, while NodeFirst takes \(|V| + 1\) operations. Thus, NodeFirst is more efficient than Zigzag.

4. STORY CONTEXT TRACKING

Transient stories we identified from the capillary network may be highly correlated, e.g., “the launch of Blackberry 10” and “Blackberry Super Bowl ad”. In this section, we exploit and integrate signals from different measures in story correlation computation, and propose deterministic context search (DCS) to construct the veins for a story \(S\) on the fly. The performance of DCS is proportional to
the size of \( S \)'s neighboring posts. In the case when the neighboring post size is huge, we propose randomized context search (RCS) to boost the performance.

### 4.1 Story Correlation Dimensions

Recall that stories correspond to \((k, d)\)-cores in the capillary network and let \( S_i(V_i, E_i) \) and \( S_j(V_j, E_j) \) denote two stories. Here we introduce different types of correlation dimensions, which capture the story correlation from different perspectives, and quantify the correlation by a value in \([0, 1]\). Notice that node overlap is a very common evidence to assess the correlation between two subgraphs [25]. However, since Proposition 1 shows that \((k, d)\)-cores generated by Zigzag or NodeFirst are pairwise disjoint in nodes, node overlap is not useful in story correlation computation.

**Dimension 1: Content Similarity.** By viewing a story as a document and a post entity as a term, existing document similarity measures can be used to assess the story correlation. However, TF-IDF based Cosine similarity fail to be effective, since TF measures can be used to assess the story correlation. However, document and a post entity as a term, existing document similarity

Dimension 1: Content Similarity. By viewing a story as a document and a post entity as a term, existing document similarity measures can be used to assess the story correlation. However, TF-IDF based Cosine similarity fail to be effective, since TF vectors of stories tend to be very sparse. We exploit another popular measure, LDA-based symmetric KL-divergence [20]. Supposing \( d \) is the document representation of story \( S \) and \( \theta(d) \) is the topic distribution of \( d \) produced by LDA, we have

\[
C_1(S_i, S_j) = \frac{KL(\theta(d_i) \| \theta(d_j)) + KL(\theta(d_j) \| \theta(d_i))}{2}
\]

where \( KL(\theta(d_i) \| \theta(d_j)) = \sum_d \theta_d(d_i) \log \frac{\theta_d(d_i)}{\theta_d(d_j)} \).

**Dimension 2: Temporal Proximity.** Stories that happen closer in time are more likely to correlate together. Given a story \( S_i(V_i, E_i) \), we can get the histogram of post volume along time dimension on each bin with size equal to the time window sliding step \( \Delta t \). Supposing \( \text{Dist} \) is the normalized distribution (with the sum equal to 1) of story \( S \)'s histogram, we take the complement of \( L_1 \) Norm as an example to measure the temporal proximity between \( S_i \) and \( S_j \):

\[
C_2(S_i, S_j) = 1 - \sum_{t=1}^{\text{len}} |\text{Dist}_i(t) - \text{Dist}_j(t)|
\]

where \( \text{len} \) is the length of the observation time window.

**Dimension 3: Edge Connectivity.** Capillaries associated with posts of two stories can be used to determine the story correlation. This approach calculates the strength of edge connectivity between posts of \( S_i \) and \( S_j \) in capillary network. These edges serve as the bridge between two stories, and the connectivity strength can be measured by various ways, e.g., the Jaccard Coefficient based on the portion of bridge edges:

\[
C_3(S_i, S_j) = \frac{|E(V_i, V_j)|}{|E(V_i, V_j) \cup E(V_j, V_i)|}
\]

where \( E(A, B) \) is the edge set between node set \( A \) and \( B \).

**Baseline: Hybrid Correlation Model.** As a baseline approach, story correlation can be computed by a hybrid model on all correlation dimensions, with the form:

\[
Cor_H(S_i, S_j) = \frac{3}{k=1} C_k(S_i, S_j)
\]

It is easy to see that \( 0 \leq Cor_H(S_i, S_j) \leq 1 \). One drawback of the hybrid correlation model is the performance. First, LDA computation for large corpus is expensive [26]. Second, to construct the context of story \( S \) in the story vein, the hybrid correlation model needs to compute \( Cor_H(S, S') \) between \( S \) and every other story \( S' \) in \( G(V, E) \), which is redundant and time-consuming.

### 4.2 Story Context Search

Story context search aims at finding the neighbors of a story in the story vein \( G(V, E) \) efficiently. However, simply applying the hybrid correlation model shown in Eq.(5) results in an inefficient pairwise computation. To overcome the challenge, we introduce iceberg query on story vein, as stated below.

**Definition 4. (Iceberg Query)** Given a story vein \( G(V, E) \), a threshold \( \gamma \) (\( 0 < \gamma < 1 \)) and a story \( S \) (\( S \notin V \)), the iceberg query for \( S \) is to find the subset of all stories \( V_S \subseteq \nabla S \), where for each \( S' \in V_S \), \( \text{Cor}(S, S') \geq \gamma \).

Iceberg query is the key technique to support CAST. Iceberg query grows the story vein on the fly: for each new story \( S \), it can quickly build possible correlation links between \( S \) and the remaining stories. Iceberg query does not need to explore all stories in \( \nabla \) to generate the exact \( V_S \), which improves the performance.

In this section, we propose two approaches to implement iceberg query on story vein. The first approach is deterministic, which integrates content similarity, temporal proximity and edge connectivity together by aggregating all capillaries between two stories. The second approach is randomized, which improves the performance of the deterministic approach by filtering the flow from \( S \) to \( S' \). They are discussed separately below.

**Deterministic Context Search.** The basic idea of deterministic context search (DCS) follows a propagation and aggregation process. To initialize, we treat story \( S(V_S, E_S) \) as source and every other story \( S'(V_{S'}, E_{S'}) \) as target. In the propagation phase, we start from each post in source and broadcast the post similarity to neighboring posts along capillaries. In the aggregation phase, we aggregate the post similarity received by posts in each target, and denote it by \( P_AD(S, S') \). In detail, we have

\[
PAD(S, S') = \sum_{p \in V_S, p' \in V_{S'}, (p, p') \in E} s(p, p')
\]

Since \( s(p, p') \) is computed by combining content similarity and temporal proximity (see Eq.(1)), \( P_AD(S, S') \) naturally integrates the above discussed three story correlation dimensions. The correlation between \( S \) and \( S' \) is assessed by the ratio between the post similarity aggregated by \( S' \) and propagated by \( S \):

\[
Cor_D(S, S') = \frac{P_AD(S, S')}{P_AD(S, G)}
\]

where \( P_AD(S, G) \) is the post similarity sum of all capillaries associated with posts in \( S \). When \( Cor_D(S, S') \geq \gamma \), there will be a story link between \( S \) and \( S' \) in story vein \( G(V, E) \). Let \( \sigma' \) denote the average time stamp of \( S' \), i.e., \( S' = \frac{1}{\sigma'} \sum_{p \in V_{S'}} p' \). If \( S' \prec S'' \), the edge direction is from \( S \) to \( S' \). Otherwise, the edge direction is from \( S' \) to \( S \).

The sketch of DCS is shown in Algorithm 3. Notice that post similarity is not aggregated on every story. Some stories may receive zero or little post similarity, and can be omitted. On average, a source story visits at most \( \min(|V_S|, \frac{|V_{S'}|}{|E|}) \) neighboring posts in DCS. Compared with the hybrid correlation model, which requires to access the whole capillary network \( G(V, E) \) in pairwise computation, DCS improves the performance significantly.

**Randomized Context Search.** To improve the performance of the deterministic context search further, we propose randomized context search (RCS) based on random walk. The motivation behind RCS is, in the propagation phase, it is preferable to only visit the neighboring posts with a high post similarity. Given a post \( p \) in story \( S \), suppose \( \max(p) \) and \( \min(p) \) are the maximum and minimum
Algorithm 3: Deterministic Context Search (DCS)

Input: \( G(V,E), \mathbb{G}(V,E) \), new story \( S(V_S, E_S) \), \( \gamma \)
Output: In-neighbor set \( N_I(S) \), out-neighbor set \( N_O(S) \)
1. \( \mathcal{P}A_D(S,G) = 0 \), \( N(S) = \emptyset \), \( N_I(S) = \emptyset \), \( N_O(S) = \emptyset \);
2. \( S'^t = \frac{1}{\|V_S\|} \sum_{p \in V_S} p' \);
3. for each \( p \in V_S \) do
   4. for each neighbor \( p' \) of \( p \) in \( G(V,E) \) do
      5. if \( p' \) is in a story \( S'R \) then
         6. if \( S' \notin N(S) \) then
             7. add \( S' \) into \( N(S) \);
             8. add \( s(p,p') \) into \( \mathcal{P}A_D(S,G) \) and \( \mathcal{P}A_D(S,S') \);
   9. for each \( S'' \in N(S) \) do
   10. \( \text{Cor}_D(S', S'') = \frac{\mathcal{P}A_D(S', S'')}{\mathcal{P}A_D(S,G)} \); 
   11. if \( \text{Cor}_D(S', S'') \geq \gamma \) then
       12. if \( S'^t < S''^t \) then add \( S'' \) into \( N_O(S) \);
       13. else add \( S'' \) into \( N_I(S) \);
15. return \( N_I(S) \) and \( N_O(S) \);

Algorithm 4: Randomized Context Search (RCS)

Input: \( G(V,E), \mathbb{G}(V,E) \), new story \( S(V_S, E_S) \), \( \gamma, \alpha, n \)
Output: In-neighbor set \( N_I(S) \), out-neighbor set \( N_O(S) \)
1. \( \mathcal{P}A_R(S,G) = 0 \), \( N(S) = \emptyset \), \( N_I(S) = \emptyset \), \( N_O(S) = \emptyset \), \( \text{count} = 0 \);
2. \( S'^t = \frac{1}{\|V_S\|} \sum_{p \in V_S} p' \);
3. while \( \text{count} < n \) do
   4. randomly select a node \( p \in V_S \);
   5. randomly select a neighbor \( p' \) of \( p \) in \( G(V,E) \);
   6. while \( s(p,p') < \min(p) + \alpha (\max(p) - \min(p)) \) do
      7. randomly select a neighbor \( p'' \) of \( p \) in \( G(V,E) \);
   9. if \( p'' \) is in a story \( S'R \) then
      10. if \( S'' \notin N(S) \) then
          11. add \( S'' \) into \( N(S) \);
          12. add \( s(p,p'') \) into \( \mathcal{P}A_R(S,G) \) and \( \mathcal{P}A_R(S,S'') \);
   13. for each \( S'' \in N(S) \) do
   14. \( \text{Cor}_R(S', S'') = \frac{\mathcal{P}A_R(S', S'')}{\mathcal{P}A_R(S,G)} \); 
   15. if \( \text{Cor}_R(S', S'') \geq \gamma \) then
       16. if \( S'^t < S''^t \) then add \( S'' \) into \( N_O(S) \);
       17. else add \( S'' \) into \( N_I(S) \);
18. return \( N_I(S) \) and \( N_O(S) \);

post similarity associated with \( p \), respectively. Technically, we use \( \alpha (0 \leq \alpha \leq 1) \) as a throttle to propagate high post similarities: only if \( s(p,p') \geq \min(p) + \alpha (\max(p) - \min(p)) \), RCS propagates \( s(p, p') \) to \( p' \) by a uniformly random probability.

There are two special cases in RCS:
- \( \alpha = 0 \): Post \( p \) propagates to every neighboring post.
- \( \alpha = 1 \): Post \( p \) only propagates to the neighboring post with the highest post similarity.

We empirically choose \( \alpha = 0.5 \). Suppose \( N \) is the total number of simulations we run for the source story \( S \). On each simulation, a random surfer starts from a post \( p \) in \( S \) and randomly visits a neighbor \( p' \) if \( s(p, p') \geq \min(p) + \alpha (\max(p) - \min(p)) \). The aggregated post similarity from source \( S \) to target \( S' \) on simulation \( i \) can be described as

\[
R_i(S, S') = \begin{cases} 
   s(p,p') & \text{if } p \in V_S, p' \in V_{S'} \\
   0 & \text{otherwise} 
\end{cases} 
\] (8)

The aggregated post similarity by \( S' \) on \( N \) simulations is denoted by \( \mathcal{P}A_R(S, S') \):

\[
\mathcal{P}A_R(S, S') = \sum_{i=1}^{N} R_i(S, S') 
\] (9)

We show the sketch of RCS in Algorithm 4. Similar to DCS, the correlation between \( S \) and \( S' \) is computed by \( \text{Cor}_R(S', S'') = \frac{\mathcal{P}A_R(S', S'')}{\mathcal{P}A_R(S,G)} \). The vein threshold and direction is determined by the same way of DCS. Clearly, in RCS, a source story visits at most \( N \) neighboring posts and usually \( N \ll |V| \). The value of \( N \) is decided by the time budget of story context search in real CME system. Compared with DCS, RCS achieves better performance by accessing smaller number of neighboring posts connected by capillaries with higher post similarity.

4.3 Interpretation of Story Vein

Since story vein is described in graph notation, it becomes very important to explain the story vein to end users who may be non-technical. There are various options to present a \((k,d)\)-Core in human consumable form. The first option is annotating a \((k,d)\)-Core by the most representative posts inside, e.g., posts with the highest edge degree. The second option is rendering a \((k,d)\)-Core into a

word cloud. Since we extracted entities from each post, the frequency of an entity in a \((k,d)\)-Core can be used to indicate the font size of that entity in the word cloud. In this paper, we provide both options to aid human perception. We will show some interpretation examples in experimental study.

5. EXPERIMENTAL STUDY

All experiments are conducted on a computer with Intel 2.66 GHz CPU, 8 GB RAM, running 64-bit Windows 7. All algorithms are implemented in Java.

Parameter Setting. We empirically set the threshold \( \epsilon = 0.2 \) for the construction of capillary network and story vein. We set \( \alpha = 0.5 \) to filter out capillaries with low post similarity in RCS.

5.1 Data Sets

All data sets are crawled from Twitter.com via Twitter4J\(^2\) API. Although our story teller CAST works regardless of the domain, we make the data set domain-specific in order to facilitate evaluation of the generated results.

CNN-News. By simulating a social news stream, we collect tweets created in the first half year of 2014 from CNN channels, which include @cnn, @cnnbrk, @cnnmoney, @cnnlive and @cnni. This data set has 10872 tweets and serves for the quality analysis.

Tech-Lite. We built a technology domain dataset called Tech-Lite by aggregating timelines of users listed in Technology category of “Who to follow”\(^3\) and their retweeted users. Tech-Lite has 352,328 tweets, 1402 users and the streaming rate is 11700 tweets/day.

Tech-Full. Based on the intuition that users followed by users in Technology category are most likely also in the technology domain, we obtained a larger technology social stream called Tech-Full by collecting all the timelines of users that are followed by users in Technology category. Tech-Full has 5,196,086 tweets, created by 224,242 users, whose streaming rate is about 7216 tweets/hour.


---

\(^2\)Twitter4J. http://twitter4j.org/

\(^3\)http://twitter.com/who_to_follow/interests
Table 2: The number of edges in the capillary network, and the number of stories and correlation links in the story vein as the changing of temporal proximity functions. We define a story as a (5, 3)-Core.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Capillaries</th>
<th>Stories</th>
<th>Vein Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>No-Fading</td>
<td>115936</td>
<td>373</td>
<td>495</td>
</tr>
<tr>
<td>Exp-Fading</td>
<td>327390</td>
<td>87</td>
<td>275</td>
</tr>
<tr>
<td>Reci-Fading</td>
<td>357132</td>
<td>123</td>
<td>384</td>
</tr>
</tbody>
</table>

Figure 3: Top 10 results of HashtagPeaks, MentionPeaks and EntityPeaks on Tech-Lite dataset. The ground truth for precision and recall is top 10 major stories selected from main stream technology news websites.

5.2 Tuning Capillary Network

Post similarity computation shown in Eq. (1) influences the structure of capillary network directly. We can tune the content similarity function $s_T(p_i^t, p_j^t)$ and temporal proximity function $s_T(|p_i^t - p_j^t|)$ to make the capillary network concise and expressive. Many set-based similarity measures such as Jaccard coefficient [17] can be used to compute the similarity $s_T(p_i^t, p_j^t)$ between posts. Since entity usually appears once in one tweet, similarity measures such as Cosine similarity and Pearson correlation [17] will degenerate to a form very similar to Jaccard, so we use Jaccard as our similarity function and omit further discussion of alternatives.

Temporal proximity function $s_T(|p_i^t - p_j^t|)$ determines how similarity to older posts is penalized, compared to recent posts. We compared three different functions: (1) reciprocal fading (“Reci-Fading”) with $D(|p_i^t - p_j^t|) = \frac{1}{|p_i^t - p_j^t|+1}$, (2) exponential fading (“Exp-Fading”) with $D(|p_i^t - p_j^t|) = e^{-|p_i^t - p_j^t|}$ and (3) no fading (“No-Fading”) with $D(|p_i^t - p_j^t|) = 1$. For any posts $p_i, p_j$, clearly $e^{\frac{1}{|p_i^t - p_j^t|}} \geq |p_i^t - p_j^t| + 1 \geq 1$. Exp-Fading penalizes the posts in the old part of time window severely (see Table 2): the number of capillaries and stories generated by Exp-Fading is lower than by other approaches. Since No-Fading does not penalize old posts in the time window, too many capillaries and stories will be generated without considering recency. Reci-Fading is in between, which is a more gradual penalty function than Exp-Fading. In the rest of experiments, we use Exp-Fading, since it generates the most concise capillary network with an emphasis on recent posts.

5.3 Quality Evaluation

In this section, we evaluate the quality of CAST on two tasks: story discovery and context search, by comparing with baselines.

Ground Truth. We build the ground truth of tech stories in Jan 2012 by browsing news articles on main stream technology websites like CNET.com, TechCrunch.com, and ReadWrite.com, etc. We pick headlines with high occurrence and build a ground truth of top 10 stories. They include CES related stories, SOPA & PIPA related stories, Facebook IPO, Yahoo co-founder Jerry Yang resignation, RIM CEO changes, etc.

Baseline 1: Peak Detection. Some recent work on event detection [14, 18, 23] can be used for story discovery, and they share the same spirit that aggregates the frequency of topic-indicating phrases at each moment to build a histogram and generates stories by detecting volume peaks in the histogram. We design three baselines to capture the major techniques used by these approaches.

- **HashtagPeaks**: aggregate frequent hashtags;
- **MentionPeaks**: aggregate frequent mentions;
- **EntityPeaks**: aggregate frequent entities.

We show top 10 results of HashtagPeaks, MentionPeaks and EntityPeaks in Tech-Lite dataset with time stamps between Jan 1 and Feb 1, 2012 in Figure 3. Their precision and recall are also listed by comparing with the ground truth. Notice that multiple peaks may correspond to the same story in ground truth, where the precision and recall differ. As we can see, MentionPeaks is the worst since entities are extracted from the social stream preprocessing stage to annotate stories. Although these baselines are able to indicate some words about a story, they’re not qualified for a successful story-teller because the internal and external structure of the story is missing. In particular, they cannot show how users interact with each other, how posts are clustered, and how the story is distributed along time dimension.

Baseline 2: Topic Modeling. Topic modeling is a typical way to detect topics from text document corpus. As we mentioned earlier, existing work on topic detection and tracking falls into a classification problem, and prevalent topic modeling models like Latent Dirichlet Allocation (LDA) [6] are mainly trained on formal writing news articles. We design a baseline using LDA, and treat top 100 posts of each topic as a story. Post time stamps are totally ignored in LDA. We set the topic number as 50, and after rendering topics to stories, we show the top 20 stories in Table 3. As we can see, the topic cohesion of LDA-detected stories is not very high: posts sharing some common words are very easily classified into the same topic, even though they are not exactly talking about the same story. Besides, LDA model cannot deal with noisy posts, so the quality is compromised in social streams.

CAST on Tech-Lite. Recall that our proposed story-teller CAST uses a $(k, d)$-Core to represent a story, which is a cohesive subgraph in the capillary network. Figure 4 shows top 10 transient stories generated by CAST. Each story is represented as a (5,3)-Core. It is worth noting that there are various ways to present stories on the user interface. In this experiment, we render a story into an entity cloud. Each cloud is annotated by a short description beside. Interestingly, we observe the curve of tweet frequency goes down on every weekend, which reflects the living habit in reality. Compared with the ground truth, our story-teller achieves a precision of 0.9 and a recall of 0.8. For precision, the only case we fail is that “hug/hope new year” is not a story in ground truth. The reason may be lots of people tweeting about “happy new year” but this is not formally reported in technology news websites, since it is a well-known fact. For recall, the ground truth story not appearing in our top 10 is “Apple iBooks 2 Release” on Jan 19. This story is ranked on the 13th position in our story-teller.
Figure 4: Top 10 transient stories generated by our proposed story-teller on Tech-Lite. Each story is represented as a (5,3)-Core, and we render a story into an entity cloud for human reading. Some transient stories may be correlated, as linked by lines. The curve on the bottom is the breakdown of tweet frequency on each day in Jan 2012.

<table>
<thead>
<tr>
<th>Rank</th>
<th>LDA-Detected Story Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>blog post, great buy in january, start sign</td>
</tr>
<tr>
<td>2</td>
<td>report amazon kindle fire</td>
</tr>
<tr>
<td>3</td>
<td>check real youtube interview video series</td>
</tr>
<tr>
<td>4</td>
<td>win chance south beach trip</td>
</tr>
<tr>
<td>5</td>
<td>small business company story, local community</td>
</tr>
<tr>
<td>6</td>
<td>make great start, things learn</td>
</tr>
<tr>
<td>7</td>
<td>bad thing in life, feel good</td>
</tr>
<tr>
<td>8</td>
<td>send email hey, glad to hear, follow tweet</td>
</tr>
<tr>
<td>9</td>
<td>jan travel, days ago, back home tonight</td>
</tr>
<tr>
<td>10</td>
<td>president obama, iran war killed</td>
</tr>
<tr>
<td>11</td>
<td>daily top stories, tech news, health care</td>
</tr>
<tr>
<td>12</td>
<td>apple tv iphone ipad, ces event, app store</td>
</tr>
<tr>
<td>13</td>
<td>watch night show, tonight fun, miss baby</td>
</tr>
<tr>
<td>14</td>
<td>twitter mobile app, android apps, tweet free</td>
</tr>
<tr>
<td>15</td>
<td>iowa caucus results, santorum and romney</td>
</tr>
<tr>
<td>16</td>
<td>play half game, nyc new york city, winter fans</td>
</tr>
<tr>
<td>17</td>
<td>super bowl, man football party</td>
</tr>
<tr>
<td>18</td>
<td>happy new year, hope wonderful year, friends family</td>
</tr>
<tr>
<td>19</td>
<td>love word, perfect friend, people</td>
</tr>
<tr>
<td>20</td>
<td>google internet search ad, https link</td>
</tr>
</tbody>
</table>

Table 3: Top 20 stories detected by LDA from Tech-Full and described by high frequency words. We treat top 100 posts of each topic as a story.

We can see that stories may be correlated with each other. In Figure 4, there are multiple stories about CES (Consumer Electronic Show) 2012, held from Jan 8 to Jan 13. Meanwhile, stories related to SOPA & PIPA are highly correlated, even though each of them tells a relatively different story.

When inspecting the top 50 transient stories returned by CAST, we observe two major advantages compared with news reports. First, our story-teller can identify stories that are hardly noticeable in news websites. For example, “Rupert Murdoch joins Twitter” is a story widely spread in social streams, but not commonly reported by news articles. Second, the formation of a transient story in social streams generally occurs earlier than the first publishing of the corresponding news articles. [14] also observed this phenomenon, and the typical time lag is around 2.5 hours.

CAST on Tech-Full. Figure 6 shows an example to illustrate the story vein on Tech-Full. To help understand, we group stories densely connected by vein links in rectangles, where each rectangle can be viewed as a story series. As we can see, “CES” and “SOPA & PIPA” are two very different story series, but they can be connected by indirect links via posts such as “CES 2012: Microsoft Keynote With Steve Ballmer” and “Microsoft opposes SOPA”. CAST will track the correlation between stories and build a highly informative story vein for story-teller.

CAST on CNN-News. CNN-News simulates the daily social stream received by a Twitter user and has a time span of six months. We show a fragment of the story vein tracked from CNN-News in Figure 5. In this example, “MH370 lost and search” is a story series with an internal structure, by organizing individual stories together. Users can click on each story to see the details, or traverse along story veins to read highly related stories happening earlier or later. Compared with Twitter Trends Search which shows trending top-ics, CAST has distinct advantages on providing both the internal structure and the external context of a specific story.

5.4 Performance Testing
In this section, we test the performance of our proposed algorithms. All tests are performed on Tech-Full dataset with the time window set to one week. On average, the capillary network has a node size 710,000 and edge size 4,020,000. Story Identification. Recall that both k-Core and (k, d)-Core generations have polynomial time complexity. We test the running time of generating all maximal k-Cores and (k, d)-Cores from the capillary network and show the result in Figure 7(a). Let k = d + 1. As d increases, the running time of k-Core generation drops. The reason is, although a bigger k means more nodes need to be removed at each iteration, the total number of iterations drops even more quickly. The running time for (k, d)-Cores is nearly stable, but much higher than k-Core generation. This observation implies that NodeFirst should be much faster than Zigzag, since NodeFirst performs k-Core generation whenever possible. This conclusion is supported by the experimental results in Figure 7(a).

https://twitter.com/search-home
The performance of RCS is remarkably better than the performance of iceberg queries decrease. In experiments, we can see that the size of transient stories smaller and naturally the total running time of RCs decrease gradually from Figure 7(b). This will make the total running time of iceberg queries decrease. In experiments, we can see that the performance of RCS is remarkably better than the performance of iceberg queries.

All experiments are running on Tech-Full dataset with the time window set to one week.

In Figure 6, we illustrate our context-aware story-teller. Each tag cloud here is a single story identified from the capillary network. Sets of stories with higher correlation are grouped together in rectangles to aid readability.

The relationship between d and the number of connected components we can find from the capillary network is discovered in Figure 7(b). We see that with increasing d, both the numbers of (d + 1)-Cores and (d + 1, d)-Cores drop. Since a (d + 1, d)-Core is at least a (d + 1)-Core but more cohesive, we may find multiple (d + 1, d)-Cores from a (d + 1)-Core by removing edges that do not satisfy the (k, d)-Core definition. Thus, given the same capillary network, the number of (d + 1, d)-Cores is usually higher than the number of (d + 1)-Cores.

**Iceberg Queries.** In CAST, we use iceberg query to construct possible vein links between a given story and all other stories in the capillary network. As discussed in Section 4, we treat the given story as source and all other stories as targets, and the story correlation threshold γ will govern the construction of story vein links.

In Figure 7(c), we treat each story in the capillary network as the query and show the total running time of iceberg queries for all stories. As d increases, we observe that the number of (d + 1, d)-Cores decrease gradually from Figure 7(b). This will make the total size of transient stories smaller and naturally the total running time of iceberg queries decrease. In experiments, we can see that the performance of RCS is remarkably better than the performance of DCS. For the quality, we show the accuracy of RCS in Table 4, as the number of simulations n grow. We define n as a number proportional to the neighboring post size of story S and measure the accuracy based on DCS.

### Table 4: The accuracy of RCS, as the number of simulations n grow

<table>
<thead>
<tr>
<th>n</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg(1 − Cor(S, S′)) / Cor(S, S′)</td>
<td>0.91</td>
<td>0.95</td>
<td>0.98</td>
<td>0.99</td>
</tr>
</tbody>
</table>

6. RELATED WORK

The related work of this paper falls into one of the following four categories.

**Social Stream Mining.** Mining social streams has gained popularity in recent years due to the rise of social media like Twitter and Facebook. The most relevant work in this category is event detection. Weng et al. [28] build signals for individual words by applying wavelet analysis on the frequency-based signals of words to detect events from Twitter. A framework for tracking short, distinctive phrases (called “memes”) that travel relatively intact through on-line text was developed in [14]. Twitinfo [18] represents an event it discovers from Twitter by a timeline of related tweets. [15] investigated the real-time interaction of events such as earthquakes in Twitter and proposed an algorithm to monitor tweets and to detect a target event based on classifiers. Compared with the above, our approach represents a transient story as a cohesive subgraph in post network, which has a compact internal structure to describe the same story. [23] and [3] take a different approach by representing an event as a small weighted graph of entities. One drawback is that since entity occurrences in multiple posts are aggregated into a single entity node and post similarities are aggregated numerically on edge weights, post attributes like time stamps cannot be reflected on entity graphs.

**Topic Detection and Tracking.** Topic detection and tracking is an extensively studied field [16], with the most common approaches based on Latent Dirichlet Allocation (LDA) [6]. Techniques on topic detection and tracking cannot be applied to story correlation tracking, because they are usually formulated as a classification problem [2], with an assumption that topics are predefined before tracking, which is unrealistic for social streams. Recent works [10, 8] prey to this problem. Besides, the lack of training data set for
story correlation tracking on noisy social streams renders the existing works [20, 24] on Story Link Detection (SLD) inapplicable, because SLD is trained on well-written news articles.

**Cohesive Subgraph.** There are several choices for clique relaxations, e.g., quasi-clique, $k$-Core, $k$-Plex, etc [13]. Out of them, finding the maximal quasi-clique and $k$-Plex are NP-Hard problems [7, 4] and furthermore, the inapproximability result of cliques [9] carries over to maximal quasi-cliques and $k$-Plexes. $k$-Cores can be computed in polynomial time, but in terms of semantics, they lack evidence to show that every two nodes in a $k$-Core are very likely describing the same story. In this paper, we define a new type of cohesive subgraph called $(k, d)$-Core for better capture of stories.

**Correlation Computation.** The correlation between nodes in a graph is a well-studied problem, with popular algorithms such as HITS, Katz and Personalized PageRank [5]. For the correlation between cohesive subgraphs defined as $(k, d)$-Cores, traditional measures like Jaccard coefficient, Cosine similarity and Pearson’s Correlation do not overlap on node sets. Recently, a propagation and aggregation process can be used to simulate the information flow between nodes, which was studied in the context of top-$k$ structural similarity search in [12] and authority ranking in [15]. In this paper, we propose our context search for iceberg query based on the propagation and aggregation process on the subgraph level.

7. CONCLUSION

In this paper, we focus on two problems: (1) Efficiently identify transient stories from fast streaming social content; (2) Perform iceberg query to build the structural context between stories. To solve the first problem, we transform social stream in a time window to a capillary network, and model transient stories as $(k, d)$-Cores in the capillary network. Two polynomial time algorithms are proposed to extract maximal $(k, d)$-Cores. For the second problem, we propose deterministic context search and randomized context search to support the iceberg query, which allows to perform context search without pairwise comparison. We perform detailed experimental study on real Twitter streams and the results demonstrate the effectiveness and value of our proposed context-aware story-teller CAST. In future work, we are interested in mining opinions from transient stories, e.g., the sentiment on a political event. Besides, we are interested in various visualization techniques to present transient stories to end users in a friendly way.

8. REFERENCES