Story Forest: Extracting Events and Telling Stories from Breaking News

BANG LIU, University of Alberta, Canada
FRED X. HAN, University of Alberta, Canada
DI NIU, University of Alberta, Canada
LINGLONG KONG, University of Alberta, Canada
KUNFENG LAI, Tencent, China
YU XU, Tencent, China

Extracting events accurately from vast news corpora and organize events logically is critical for news apps and search engines, which aim to organize news information collected from the Internet and present it to users in the most sensible forms. Intuitively speaking, an event is a group of news documents that report the same news incident possibly in different ways. In this paper, we describe our experience of implementing a news content organization system at Tencent to discover events from vast streams of breaking news and to evolve news story structures in an online fashion. Our real-world system faces unique challenges in contrast to previous studies on topic detection and tracking (TDT) and event timeline or graph generation, in that we 1) need to accurately and quickly extract distinguishable events from massive streams of long text documents, and 2) must develop the structures of event stories in an online manner, in order to guarantee a consistent user viewing experience. In solving these challenges, we propose *Story Forest*, a set of online schemes that automatically clusters streaming documents into events, while connecting related events in growing trees to tell evolving stories. A core novelty of our *Story Forest* system is *EventX*, a semi-supervised scheme to extract events from massive Internet news corpora. *EventX* relies on a two-layered, graph-based clustering procedure to group documents into fine-grained events. We conducted extensive evaluations based on 1) 60 GB of real-world Chinese news data, 2) a large Chinese Internet news dataset that contains 11,748 news articles with truth event labels, and 3) the 20 News Groups English dataset, through detailed pilot user experience studies. The results demonstrate the superior capabilities of *Story Forest* to accurately identify events and organize news text into a logical structure that is appealing to human readers.

CCS Concepts: • **Information systems → Clustering**; **Data stream mining**; *Document topic models*; • **Computing methodologies → Natural language processing**; **Semi-supervised learning settings**;

Additional Key Words and Phrases: Story Forest, EventX, Document clustering, News articles organization, Community detection

Authors' addresses: Bang Liu, University of Alberta, Electrical and Computer Engineering, Edmonton, Canada, bang3@ualberta.ca; Fred X. Han, University of Alberta, Electrical and Computer Engineering, Edmonton, Canada, xuefei1@ualberta.ca; Di Niu, University of Alberta, Electrical and Computer Engineering, Edmonton, Canada, dniu@ualberta.ca; Linglong Kong, University of Alberta, Mathematical and Statistical Sciences, Edmonton, Canada, lkong@ualberta.ca; Kunfeng Lai, Tencent, Platform and Content (Business) Group, Shenzhen, China, calvinlai@tencent.com; Yu Xu, Tencent, Platform and Content (Business) Group, Shenzhen, China, henrysxu@tencent.com.

1556-4681/2020/1-ART1 $15.00
https://doi.org/10.1145/3377939

ACM Reference Format:
Bang Liu, Fred X. Han, Di Niu, Linglong Kong, Kunfeng Lai, and Yu Xu. 2020. Story Forest: Extracting Events and Telling Stories from Breaking News. *ACM Trans. Knowl. Discov. Data.* 1, 1, Article 1 (January 2020), 28 pages. https://doi.org/10.1145/3377939

1 INTRODUCTION

With information explosion in a fast-paced modern society, tremendous volumes of news articles are constantly being generated on the Internet by different media providers, e.g., Yahoo! News, Tencent News, CNN, BBC, etc. In the meantime, it becomes increasingly difficult for average readers to digest the huge volumes of daily news articles, which may cover diverse topics and contain redundant or overlapping information. Many news app users have the common experience that they are overwhelmed by highly redundant information about a number of ongoing hot events, while still being unable to get information about the events they are truly interested in. Furthermore, search engines perform document retrieval from large corpora based on user-entered queries. However, they do not provide a natural way for users to view trending topics or breaking news.

An emerging alternative way to visualize news corpora without pre-specified queries is to organize and present news articles through event timelines [38, 41], event threads [24], event evolution graphs [42], or information maps [33, 34, 39]. All of these approaches require the extraction of conceptually clean events from a large number of messy news documents, which involves automated event extraction and visualization as a crucial step toward intelligent news systems. However, few existing news information organization techniques successfully achieve this goal due to several reasons:

First of all, prior research on Topic Detection and Tracking (TDT) [2] as well as text clustering [1, 18] mainly focused on grouping related documents into topics—it is much harder to cluster articles by events, where articles depicting the same event should be grouped together, since the number of events that occur daily in the real world is unpredictable. As a result, we cannot use some of the popular clustering algorithms, e.g., K-means, that require predefining the number of clusters, to extract events. In addition, the sizes of event clusters are highly skewed, because hot events may be extensively discussed by tens or even hundreds of news articles on the Internet. In contrast, regular events will be reported by only a few or even one article. These single-document events, however, constitute the majority of daily news collections, and should also be accurately discovered to appeal to the diverse interests of readers.

Second, many recently proposed event graphs or information maps try to link events in an evolution graph [42] or permitting intertwining branches in the information map [34]. However, we would like to argue that such overly complex graph structures do not make it easy for users to quickly visualize and understand news data. In fact, most *breaking news* follows a much simpler storyline. Using complex graphs to represent breaking news may complicate and even blur the story structure.

Third, most existing event time-line or event graph generation schemes are based on *offline* optimization over the entire news corpora. However, for an automated event extraction system that aids the visualization of breaking news, it is desirable to "grow" the stories in an online fashion as news articles are published, without disrupting or restructuring the previously generated storylines. On one hand, given the vast amount of daily news data, incremental and online computation will incur less computation overhead by avoiding repeated processing of older documents. On the other hand, an online scheme can deliver a consistent story development structure to users, so that users can quickly follow newly trending events.

In this paper, we propose the *Story Forest*, a novel news organization system that addresses the aforementioned challenges. To extract conceptually clean events, each of which is essentially

a cluster of news documents describing the same physical breaking news event, Story Forest incorporates a novel, semi-supervised, two-layered document clustering procedure that leverages a wide range of feature engineering and machine learning techniques, including keyword extraction, community detection, and graph-based clustering. We call this clustering procedure *EventX*. To the best of our knowledge, it is the first document clustering scheme specially tailored for event extraction among breaking news documents in the open domain.

We start with the observation that documents focusing on the same topic usually contain overlapping keywords. Therefore, in the first layer of the clustering procedure in *EventX*, we utilize a classifier trained on over 10,000 news articles to distinguish keywords from non-keywords for each document. We then apply an existing community detection algorithm onto a keyword co-occurrence graph constructed from news corpora and extract subgraphs [32] of keywords to represent topics. Each document is assigned a topic by finding out its most similar keyword subgraph. However, a keyword community or a topic is still coarse-grained and may cover many events. In the second layer of *EventX*, documents within each topic are further clustered into fine-grained events. We construct a document relationship graph within each topic, where the relationship between each pair of documents, i.e., whether they describe the same event, is predicted by a supervised document pair relationship classifier trained on carefully handcrafted features. Finally, we apply the graph-based community detection algorithm again to decompose the document relationship graph of each topic into conceptually separate events.

To enhance event visualization, our Story Forest system further groups the discovered events into stories, where each story is represented by a *tree* of interconnected events. A link between two events indicates the temporal evolution or a causal relationship between the two events. In contrast with existing story generation systems such as StoryGraph [42] and MetroMap [33], we propose an online algorithm to evolve story trees incrementally as breaking news articles arrive. Consequently, each story (called a story tree) is presented in one of several easy-to-view structures, i.e., either a linear timeline, a flat structure, or a tree with possibly multiple branches, which we believe are succinct and sufficient to represent story structures of most breaking news.

Currently, access to the related public data for event extraction and organization is extremely limited. Therefore, to facilitate evaluation and further research on the problem of event clustering and story formation for breaking news, we have created multiple datasets, with the effort of dedicated editors. First, we have created the *Chinese News Corpus* dataset which contains 60 GB of Chinese news documents collected from all major Internet news providers in China (including Tencent, Sina, WeChat, Sohu, etc.) in a 3-month period from October 1, 2016 to December 31, 2016, covering very much diversified topics in the open domain. Second, we further created the *Chinese News Events* dataset, where each article is manually labeled with the true event label and story label by editors and product managers at Tencent. It is also, to the best of our knowledge, the first Chinese dataset for event extraction evaluation. The new datasets have been made publicly available for research purposes.[1]

We evaluated the performance of Story Forest based on the Chinese News Corpus dataset, and compared our EventX news document clustering algorithm with other approaches on the Chinese News Events dataset. We also conducted a detailed and extensive pilot user experience study for (long) news document clustering and news story generation to evaluate how our system as well as several baseline schemes appeal to the habit of human readers. According to the pilot user experience study, our system outperforms multiple state-of-the-art news clustering and story generation systems, such as KeyGraph [32] and StoryGraph [42], in terms of logical validity of

[1]Our Chinese News Events dataset is currently available at:https://pan.baidu.com/s/12vWHHTD8gQLPvVftm6LQdg. For the Chinese News Corpus dataset, we are currently under the process of publishing it to the public for research purposes.

the generated story structures, as well as the conceptual cleanness of each identified event/story. Experiments show that the average time for our Java-based system to finish event clustering and story structure generation based on the daily news data is less than 30 seconds on a MacBook Pro with a 2 GHz Intel Core i7 processor, and 8 GB memory. Therefore, our system proves to be highly efficient and practical.

To summarize, we make the following contributions in this paper:

- We formally define the problem of event extraction for breaking news articles in the open domain, where the granularity of an event must conform to the physical events described by the articles and can be implicitly guided by the labeled dataset in our semi-supervised algorithms. We will describe it in more details in Sec. 2.
- We propose the *EventX* algorithm, which is a two-layered, graph-based document clustering algorithm that can perform fast event extraction from a large volume of news documents in a semi-supervised manner. Note that the main novelty of *EventX* includes a *layered* clustering scheme to separate the problem of topic discovery from that of finer-grained event extraction. Such a two-layered graph-based clustering scheme significantly improves the overall time efficiency and scalability of the algorithm, making it applicable for industry practice.
- We explore a *tree-of-events* representation for visualizing news documents. We also introduce an online algorithm to dynamically incorporate new events into the existing trees. Combining this approach with the *EventX* algorithm, we create the Story Forest system, for intelligent and efficient news story structure formation.
- We have collected and labeled a large amount of data for the study and evaluation of event extraction and story structure organization, since to our best knowledge, there is no publicly available dataset specifically dedicated to news event clustering or extraction and story formation.

Our algorithm has been successfully integrated into the hot event discovery feature of Tencent QQ browser, which is one of the most popular mobile browsers that serves over 100 million daily active users.

The remainder of this paper is organized as follows. Sec. 2 formally describes the problem of event extraction and organization from massive news data. In Sec. 3, we propose the main design of *Story Forest* system and *EventX* algorithm. In Sec. 4, we describe the *Chinese News Events* dataset collected and created specifically for evaluating event extraction algorithms. We then compare and discuss the experimental results of *EventX* and *Story Forest* among other baselines. Sec. 5 reviews the related literature. The paper is concluded in Sec. 6.

2 PROBLEM DEFINITION AND NOTATIONS

In this section, we will first describe key concepts and notations used in this paper, and formally define our problem. Then, we conduct a case study to clearly illustrate the idea of story trees.

2.1 Problem Definition

We first present the definitions of some key concepts in a bottom-up hierarchy, *event* → *story* → *topic*, to used in this paper.

Definition 2.1. Event: an event $\mathcal{E}$ is a set of news documents reporting the same piece of real-world breaking news.

Definition 2.2. Story: a story $\mathcal{S}$ is a tree of related events that report a series of evolving real-world breaking news. A directed edge from event $\mathcal{E}_1$ to $\mathcal{E}_2$ indicates a temporal development relationship or a logical connection from the breaking news event $\mathcal{E}_1$ to event $\mathcal{E}_2$.

Definition 2.3. Topic: a topic consists of a set of stories that are highly correlated or similar to each other.

In this work, we assume that two news articles are describing the same event as long as the incident they cover/report has the same time of occurrence and involves the same participating entities. This assumption is highly justifiable for breaking news articles on the Internet, which unlike fictions, are usually short, succinct and focusing on reporting the incident of a group of specific persons, organizations or other types of entities, taking actions at one or several locations. We do not consider the stance, perspective or publishing date of an article. Additionally, we do not model relationships between events, other than the weightless temporal evolution required for building story trees. This allows each event to be conceptually cleaner and more specific.

In contrast, a topic is usually determined in a subjective manner, and is usually more vague and broader compared to an event. For example, both *American presidential election* and *2016 U.S. presidential election* can be considered topics, with the second topic being a subset of the first topic. We also make the assumption that each event has a single most appropriate corresponding topic. This ensures that an event cannot appear under multiple topic clusters, which significantly simplifies the clustering procedure. This assumption is also realistic for most *breaking news* articles, where an article is usually written for timeliness to report only one real-world incident.

Each topic may contain multiple story trees, and each story tree consists of multiple logically connected events. In our work, events (instead of news documents) are the smallest atomic units. Each event is also assumed to belong to a single story and contributes partial information to the overall story. For instance, considering the topic *American presidential election*, *2016 U.S. presidential election* is a story within this topic, and *Trump and Hilary's first television debate* is an event within this story.

The boundaries between events, stories and topics do not have to be explicitly defined. In real-world applications, the labeled training dataset often implicitly captures and reflects their differences. In this way, even documents that are not related to evolving physical events can be clustered according to a specific granularity, as long as the labeled training dataset contains such kinds of samples to define the boundary. Therefore, our EventX Event Extraction algorithm defines a general framework to cluster documents into events that implicitly defined by any training dataset.

Note that prior studies on text clustering usually focus on clustering at the granularity of topics [1, 2], which are clusters of related articles. In contrast, the event extraction problem is much more challenging, because on top of clustering documents that belong to the same topic, we also need to utilize key information within each document to ensure that all documents within an event cluster are reporting the same physical event.

2.2 Notations

We now introduce some notations and describe our problem formally. Given a news document stream $D = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_t, \ldots\}$, where $\mathcal{D}_t$ is the set of news documents collected on time period t, our objective is to: a) cluster all news documents D into a set of events $E = \{\mathcal{E}_1, \ldots, \mathcal{E}_{|E|}\}$, and b) connect the extracted events to form a set of stories $S = \{\mathcal{S}_1, ..., \mathcal{S}_{|S|}\}$. Each story $\mathcal{S} = (E, L)$ contains a set of events E and a set of links L, where $L_{i,j} :=< \mathcal{E}_i, \mathcal{E}_j >$ denotes a directed link from event $\mathcal{E}_i$ to $\mathcal{E}_j$, which indicates a temporal evolution or logical connection relationship.

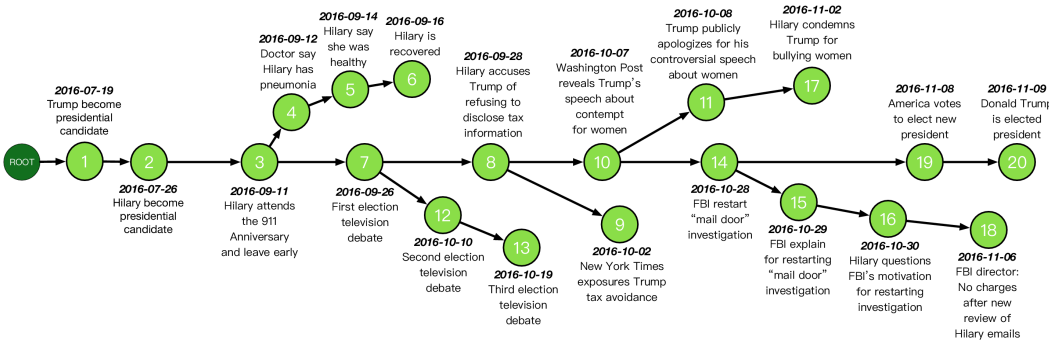


Fig. 1. The story tree of "2016 U.S. presidential election."

Furthermore, we require the events and story trees to be extracted in an online or incremental manner. That is, we extract events from each $\mathcal{D}_t$ individually when the news corpus $\mathcal{D}_t$ arrives in time period t, and *merge* the discovered events into the existing story trees that were found at time $t-1$. This is a unique strength of our scheme as compared to prior work, since we do not need to repeatedly process older documents and can deliver the complete set of evolving yet logically consistent story trees to users on-demand.

2.3 Case Study

To give readers more intuition on what the stories or events look like, here we use an illustrative example to help further clarify the concept of stories vs. events. Fig. 1 showcases the story tree of "2016 U.S. presidential election". The story contains 20 nodes, where each node indicates an event in 2016 U.S. election, and each link indicates a temporal evolution or a logical connection between two events. For example, event 19 says America votes to elect new president, and event 20 says Donald Trump is elected president. The index number on each node represents the event sequence over the timeline. There are 6 paths within this story tree, where the path $1 \rightarrow 20$ capture the whole presidential election process, branch $3 \rightarrow 6$ are about Hilary's health conditions, branch $7 \rightarrow 13$ are about television debates, $14 \rightarrow 18$ are related to the "mail door" investigation, etc. As we can see, by modeling the evolutionary and logical structure of a story into a story tree, users can easily grasp the logic of news stories and learn the main information quickly. Let us consider the following 4 events under the topic *2016 U.S. presidential election*: 1) *First election television debate*; 2) *Second election television debate*; 3) *FBI restarts "mail door" investigation*; 4) *America votes to elect the new president*. Intuitively, these 4 events should have no overlapping information between them. A news article about Trump and Hillary's *First election television debate* is conceptually separate from another article that is reporting Trump and Hillary's *Second election television debate*. For news articles, different events under the same topic should be clearly distinguishable, because they usually follow the progressing timeline of real-world affairs.

Let us represent each story by an empty root node s from which the story is originated, and denote each event by an event node e. The events in a story can be organized in one of the following four structures shown in Fig. 2: a) a flat structure that does not include dependencies between events; b) a timeline structure that organizes events by their timestamps; c) a graph structure that checks the connection between all pairs of events and maintains a subset of most strong connections; d) a tree structure to reflect the structures of evolving events within a story. Compared with a tree structure, sorting events by timestamps omits the logical connection between events, while using directed acyclic graphs to model event dependencies without considering the evolving

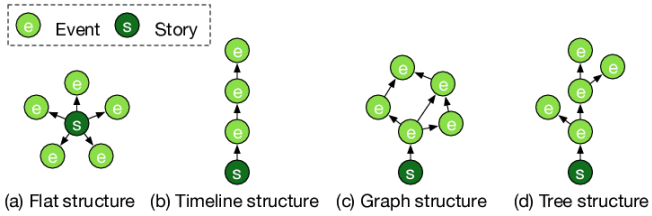


Fig. 2. Different structures to characterize a story.

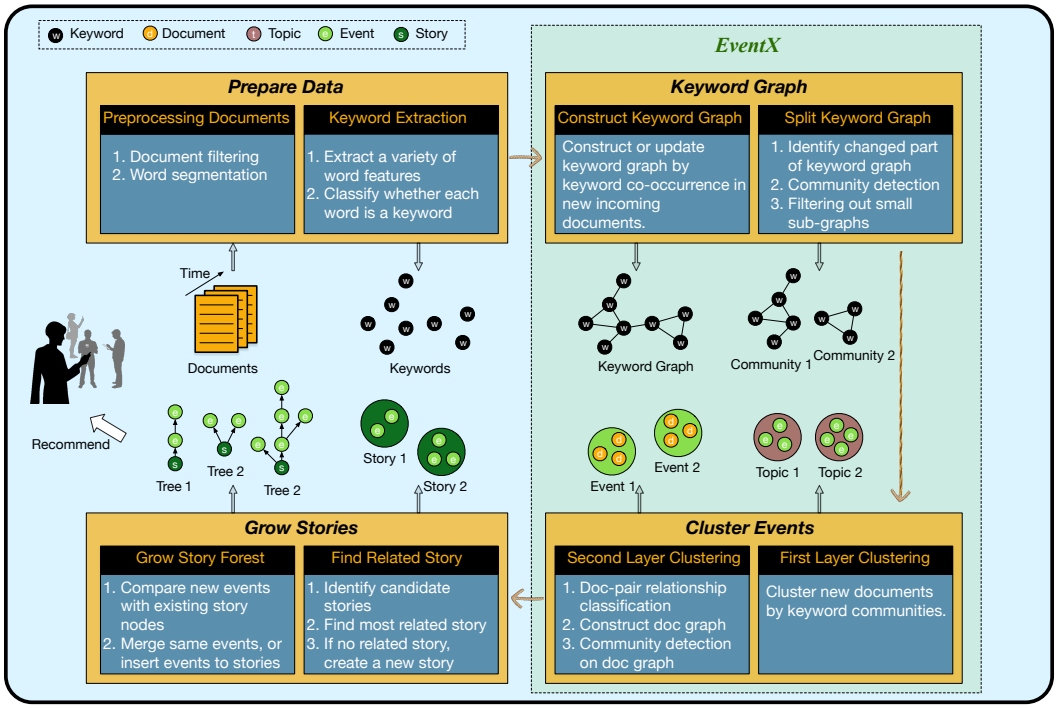


Fig. 3. An overview of the system architecture of *Story Forest*.

consistency of the whole story can lead to unnecessary connections between events. Through extensive user experience studies in Sec. 4, we show that tree structures are the most effective way to represent breaking news stories.

3 THE STORY FOREST SYSTEM

In this section, we start with an overview of the proposed *Story Forest* system. Then, we separately introduce the detailed procedures of event extraction from news documents, and how we model stories' evolutionary structure by story trees. Our implementation of the Story Forest system is open-sourced for research purpose [2].

An overview of our *Story Forest* system is shown in Fig. 3, which mainly consists of four components: preprocessing, keyword graph construction, clustering documents to events, and

[2]https://github.com/BangLiu/StoryForest

Fig. 4. The structure of keyword classifier.

growing story trees with events. The overall process is divided into 8 steps. First, the input news document stream will be processed by a variety of NLP and machine learning tools, including document filtering and word segmentation. Then we perform keyword extraction, construct/update keyword co-occurrence graph, and split the graph into sub-graphs. After that, we utilize our proposed *EventX* algorithm to cluster documents into fine-grained events. Finally, we update the story trees (formed previously) by either inserting each discovered event into an existing story tree at the right place, or creating a new story tree if the event does not belong to any existing story. Note that each topic may contain multiple story trees and each story tree consists of logically connected events. We will explain the design choices of each component in detail in the following subsections.

3.1 Preprocessing

When new documents arrive, the first task *Story Forest* performs is document preprocessing, which includes the following sequential steps:

Document filtering: documents with content length smaller than a threshold (20 characters) will be discarded.

Word segmentation: we segment the title and body of each document using Stanford Chinese Word Segmenter *Version 3.6.0* [8], which has proved to yield excellent performance on Chinese word segmentation tasks. Note that for data in a different language, the corresponding word segmentation tool in that language can be used.

Keyword extraction: extracting appropriate keywords to represent the main ideas of a document is critical to the performance of the system. We have found that traditional keyword extraction approaches, such as TF-IDF based keyword extraction and TextRank [23], cannot produce satisfying results on real-world news data. For example, the TF-IDF based method measures a word's importance by its frequency in the document. Therefore, it is unable to extract keywords that have a relatively low frequency. The TextRank algorithm utilizes the word co-occurrence information and is able to handle such cases. However, its computation time increases significantly as document length increases. Another idea is to fine-tune a rule-based system to combine multiple keyword extraction strategies. Still, such type of system heavily relies on the quality of the rules and often generalizes poorly.

To efficiently and accurately extract keywords, we trained a binary classifier to determine whether a word is a keyword to a document. In particular, we first extract the potential keywords for a document from a massive vocabulary maintained by Tencent, which contains over 10 million named entities. We regard an entity as a potential keyword if it appears in the document title or the content. We then hire human judges to determine whether a candidate keyword is indeed a keyword of the given document. Following this procedure, we collect $20,000+$ positive keywords and $350,000+$ negative keywords from $10,000+$ documents. Each keyword is transformed into a multi-view feature vector. Table 1 lists the main features that we found to be critical to the binary classifier. For the LDA feature vector, we trained a 1000-dimensional LDA model based on news data collected from January 1, 2016 to May 31, 2016 that contains $300,000+$ documents. The training process costs 30 hours.

Table 1. Features for the keyword classifier.

Type	Features
Word feature	Named entity or not, location name or not, contains angle brackets or not.
Structural feature	TF-IDF, whether appear in title, first occurrence position in document, average occurrence position in document, distance between first and last occurrence positions, average distance between word adjacent occurrences, percentage of sentences that contains the word, TextRank score.
Semantic feature	LDA

After acquiring the features for each word. A straightforward idea is to input them directly to a Logistic Regression (LR) classifier. However, as a linear classifier, Logistic Regression relies on careful feature engineering. To reduce the impact of human bias in handcrafted features, we combine a Gradient Boosting Decision Tree (GBDT) with a LR classifier to get a keyword/non-keyword classification result, as shown in Fig. 4. The GBDT component is trained to automatically discover useful cross features or combinations and discretize continuous features. The output of the GBDT will serve as the input to the LR classifier. The LR classifier will determine whether a word is a keyword for the current document. We also tried SVM as the classifier in the second stage instead of LR and observed similar performance. It is also worth mentioning that on another Chinese keyword extraction dataset, our classifier achieved a precision of 0.83 and a recall of 0.76, while they are 0.72 and 0.76 respectively if we exclude the GBDT component.

3.2 Event Extraction by EventX

After document preprocessing, we need to extract events. Event extraction here is essentially a fine-tuned document clustering procedure to group conceptually similar documents into events. Although clustering studies are often subjective in nature, we show that our carefully designed procedure can significantly improve the accuracy of event clustering, conforming to human understanding, based on a manually labeled news dataset. To handle the high accuracy requirement for long news text clustering, we propose *EventX*, a 2-layer clustering approach based on both keyword graphs and document graphs. The major steps in *EventX* include keyword co-occurrence graph construction, first-layer topic clustering based on keyword co-occurrence graph, and second-layer event extraction based on document relationship graph.

Note that the concept of “event" in this paper and our proposed dataset is fundamentally different from the “event mentions” in cross-document event coreference [19]. In our case, no matter how many event mentions can be found in an article, there is always a single major event that the news article is intended to report and cover or that triggers this report, with its narratives focusing on this major event. The task of *EventX* is to extract events by identifying whether a group of documents are reporting the same breaking news with different narratives or paraphrasing, instead of identifying what event mentions a document contains. In the following, we provide a detailed description on the inner workings of *EventX*.

3.2.1 Construct Keyword Co-occurrence Graph. Algorithm 1 shows the detailed steps of *EventX*. We show that our two-layered approach significantly improves the accuracy of event clustering on long news articles.

Algorithm 1 EventX Event Extraction Algorithm

Input: A set of news documents $\mathcal{D} = \{d_1, d_2, ..., d_{|\mathcal{D}|}\}$, with extracted features described in Sec. 3.1; a pre-trained document pair relationship classifier.

Output: A set of events $E = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{|E|}\}$.

1: Construct a keyword co-occurrence graph $\mathcal{G}$ of all documents' keywords. Connect w_i and w_j by an undirected edge $e_{i,j}$ if the times that the keywords w_i and w_j co-occur exceed a certain threshold δ_e, and $\Pr\{w_j|w_i\}$, $\Pr\{w_i|w_j\}$ are bigger than another threshold δ_p.
2: Split $\mathcal{G}$ into a set of small and strongly connected keyword communities $C = \{C_1, C_2, ..., C_{|C|}\}$, based on the community detection algorithm [25]. The algorithm keeps splitting a graph by iteratively deleting edges with high betweenness centrality score, until a stop condition is satisfied.
3: **for** each keyword community C_i, $i = 1, \ldots, |C|$ **do**
4: Retrieve a subset of documents $\mathcal{D}_i$ which is highly related to this keyword community by calculating the cosine similarity between the TF-IDF vector of each document and that of the keyword community, and comparing it to a threshold δ.
5: Connect document pairs in $\mathcal{D}_i$ to form a document relationship graph $\mathcal{G}_i^d$ using the document pair relationship classifier.
6: Split $\mathcal{G}_i^d$ into a set of document communities $C_i^d = \{C_{i,1}^d, C_{i,2}^d, ..., C_{i,|C_i^d|}^d\}$, based on the community detection algorithm. Each community represents an event.
7: **end for**

Given a news corpus $\mathcal{D}$, we construct a keyword co-occurrence graph [32] $\mathcal{G}$. Each node in $\mathcal{G}$ is a keyword w extracted by the scheme described in Sec. 3.1, and each undirected edge $e_{i,j}$ indicates that w_i and w_j have co-occurred in a same document. Edges that satisfy two conditions will remain and other edges will be pruned: the times of co-occurrence shall be above a minimum threshold δ_e (we set $\delta_e = 2$ in our experiments), and the conditional probabilities of the occurrence $\Pr\{w_j|w_i\}$ and $\Pr\{w_i|w_j\}$ also need to be greater than a predefined threshold δ_p (we use 0.15). The conditional probability $\Pr\{w_j|w_i\}$ is calculated as

$$\Pr\{w_i|w_j\} = \frac{DF_{i,j}}{DF_j}, \tag{1}$$

where $DF_{i,j}$ represents the number of documents that contain both keyword w_i and w_j, and DF_j is the document frequency of keyword w_j. It represents the probability that w_i occurs in a document if that document contains w_j.

3.2.2 Topic Clustering on Keyword Co-occurrence Graph. Next, we perform community detection on the constructed keyword co-occurrence graph. The goal is to split the whole keyword graph $\mathcal{G}$ into communities $C = \{C_1, C_2, ..., C_{|C|}\}$, where each community (or subgraph) C_i contains the keywords for a certain topic i (to which multiple events may be associated). The intuition is that keywords related to a common topic usually will appear frequently in documents belonging to that topic. For example, documents belonging to the topic "2016 U.S. presidential election" will frequently mention keywords such as "Donald Trump", "Hillary Clinton", "election" and so on. Such highly related keywords will be connected to each other in the keyword graph and form dense subgraphs, while keywords that are not highly related will have sparse connections or no connection. Our objective here is to extract dense keyword subgraphs associated with different topics.

Table 2. Features for document pair relationship classification.

Feature type	Description
Keyword	Number of common keywords in titles or contents, percentage of common keywords in titles or contents.
Textual similarity	TF-IDF (and TF) similarities of titles or contents, TF-IDF (and TF) similarities of the first N (N=1,2,3) sentences.
Semantic	LDA cosine similarity of two documents, the absolute value of the difference between the LDA vectors of the two documents.

The benefit of using community detection in the keyword graph is that it allows each keyword to appear in multiple communities. In reality, it is not unusual to have one keyword appearing under multiple topics. We also tried another method of clustering keywords by *Word2Vec*. But the performance is worse than community detection based on co-occurrence graphs. The main reason is that when clustering with pre-trained word vectors, words with similar semantic meanings are more likely to be grouped together. However, unlike articles in a specialized domain, news topics from the open domain often contain keywords with diverse semantic meanings.

To detect keyword communities, we utilize the *betweenness centrality score* [32] of edges to measure the strength of each edge in the keyword graph. The betweenness score of an edge is defined as the number of shortest paths between all pairs of nodes that pass through it. An edge between two communities is expected to have a high betweenness score. Edges with high betweenness score will be removed iteratively to divide the communities. If two edges have the same betweenness score, the one with lower conditional probability will be removed. The conditional probability of edge $e_{i,j}$ is calculated as $(\Pr\{w_i|w_j\} + \Pr\{w_j|w_i\})/2$. We calculate the betweenness score of edges using breadth first search (BFS) and iteratively remove edges with the highest betweenness score. Whenever we remove an edge, we recalculate the betweenness score of the remaining edges in the graph. As the edges is constantly removed, on one hand, the maximum betweenness score of all edges will keep decreasing. On another hand, a graph may become not fully connected and turn into two sub-graphs. Once a keyword graph is no longer fully connected, we continue the same process recursively on each newly created sub-graphs. The splitting process ends if the number of nodes in each subgraph is smaller than a predefined threshold δ_g (we use 3), or the maximum betweenness score of all edges in the subgraph is smaller than a threshold based on the subgraph's size. We refer interested readers to [32] for more details about community detection. We notice that the betweenness-based community detection algorithm is used as an off-the-shelf tool in *EventX* algorithm. There exist other community detection algorithms that are more efficient in terms of time complexity [27], and we can easily switch to these algorithms if time efficiency is a concern.

After obtaining the keyword communities, we calculate the cosine similarities between each document and each keyword community. The documents are represented as TF-IDF vectors. Given a keyword community is essentially a bag of words, it can also be considered as a document. We assign each document to the keyword community with the highest similarity, as long as the similarity is also above a predefined threshold δ. At this point, we have finished clustering in the first layer, i.e., the documents are now grouped by topics.

3.2.3 Event Extraction based on Document Relationship Graph. After we partition documents into topics, we perform the second-layer document clustering within each topic to obtain fine-grained events. We also call this process *event clustering*. An event cluster only contains documents that talk about the same event. As mentioned before, since event clusters could vary dramatically in

sizes, traditional unsupervised learning based algorithms, such as K-means, Non-negative Matrix Factorization, Hierarchical clustering, are not appropriate. Instead, we adopt a supervised-learning-guided clustering procedure in the second layer.

Specifically, in contrast with the keyword graph in the first layer, now we consider each document as a graph node, and try to connect document pairs that discuss the same event. In the keyword co-occurrence graph, judging whether two keywords are related is achieved through the co-occurrence of keywords in documents. This is feasible because the granularity of a topic is relatively coarse and subjective compared with an event. However, simply combining unsupervised strategies to judge whether two documents are talking about the same event cannot produce satisfying results, because events are highly diverse and the clustering granularity should be adjusted accordingly.

To address the above challenges, we propose a semi-supervised clustering scheme which incorporates a document pair relationship classifier to construct a document relationship graph, and performs event extraction on that graph. Given a set of documents $\mathcal{D} = \{d_1, d_2, ..., d_{|\mathcal{D}|}\}$ within a topic cluster, for each pair of documents $\langle d_i, d_j \rangle$ that belongs to $\mathcal{D}$, we add an edge $e^d_{i,j}$ if they are talking about the same event. We trained a SVM classifier to determine whether a pair of documents are talking about the same event using multiple document pair features, such as cosine similarity of TF-IDF vectors, number of common keywords, LDA cosine similarity, as the input vector. Table 2 lists the main features we utilized to train the document pair relationship classifier. For each pair of documents within the same topic, we decide whether to draw an edge between them based on the prediction made by the document pair relationship classifier. Hence, documents in each topic $\mathcal{D}$ will form a document relationship graph $\mathcal{G}^d$. We then apply the same community detection algorithm mentioned above to such graphs. Naturally, each community within a topic cluster now represents an event. Note that the graph-based clustering on the second layer is highly efficient, since the number of documents contained in each topic is significantly less after the first-layer document clustering.

In a nutshell, our two-layered scheme first groups documents into topics based on keyword community detection and then further groups the documents within each topic into fine-grained events. It has multiple advantages compared with a number of existing clustering algorithms:

(1) It does not require a pre-defined number of event clusters. This is critical in real-world applications, as for real-world data such as daily news, the exact number of events is almost always unknown and varies dramatically. Most existing clustering algorithms require the number of clusters as a parameter, or need to carefully choose parameters that control the clustering granularity. For EventX, even though we still need to specify a stopping criteria in the community detection algorithm, the threshold hyper-parameters in our algorithm are much more stable and insensitive to different sizes of news documents. Therefore, these thresholds can be selected using grid search, and do not require to be updated frequently for news articles from a different date.

(2) Our algorithm can adaptively determine the granularity of different events based on the pre-trained document pair relationship classifier, as well as community detection on the document relationship graph.

(3) The performance of our algorithm is less sensitive to the parameter settings in the first layer, because the granularity of a topic is usually subjective. We can set relatively low thresholds in the first layer to ensure that documents belonging to the same event are grouped together, then extract the fine-grained events in the second layer.

(4) The performance of event clustering can be improved by adding more features in Table 2 to represent the similarities between two documents or increasing training data to improve the document pair relationship classifier in the second-layer clustering procedure, without affecting keyword extraction and topic clustering in the first-layer clustering. We can also apply a more sophisticated algorithm for document pair relationship classification by representing two documents as a Concept Interaction Graph and apply graph convolution, as we have introduced in [20].

(5) The two-layered scheme of our algorithm makes event clustering highly efficient. Directly performing relationship classification within a document corpora is not feasible, as the time complexity is at least $O(n_d^2)$ where n_d is the number of documents. By utilizing the two-layered scheme, the documents will be split into multiple topics (each topic usually contains 1 to 100 documents). Therefore, the time complexity for the second-layer event extraction is approximately $O(n_d)$. The total time complexity that taking both two layers of clustering into account is still much smaller than $O(n_d^2)$, as we will discuss in Sec. 4.4

3.3 Growing Story Trees Online

Given the set of extracted events for a particular topic, we further organize these events into multiple stories under this topic in an online manner. Each story is represented by a *Story Tree* to characterize the evolving structure of that story. Upon the arrival of a new event and given an existing story forest, our online algorithm to grow the story forest mainly involves two steps: a) identifying the story tree to which the event belongs; b) updating the found story tree by inserting the new event at the right place. If this event does not belong to any existing story, we create a new story tree.

a) Identifying the related story tree. Given a set of new events $E_t = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{|E_t|}\}$ at time period t and an existing story forest $\mathcal{F}_{t-1} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_{|\mathcal{F}_{t-1}|}\}$ that has been formed during previous $t-1$ time periods, our objective is to assign each new event $\mathcal{E} \in E_t$ to an existing story tree $\mathcal{S} \in \mathcal{F}_{t-1}$. If no story in the current story forest matches that event, a new story tree will be created and added to the story forest.

We apply a two-step strategy to decide whether a new event $\mathcal{E}$ belongs to an existing story tree $\mathcal{S}$ formed previously. First, as described at the end of Sec. 3.2, event $\mathcal{E}$ has its own keyword set $C_\mathcal{E}$. Similarly, for the existing story tree $\mathcal{S}$, there is an associated keyword set $C_\mathcal{S}$ that is a union of all the keyword sets of the events in that tree.

Then, we can calculate the compatibility between event $\mathcal{E}$ and story tree $\mathcal{S}$ as the Jaccard similarity coefficient between $C_\mathcal{S}$ and $C_\mathcal{E}$: $\text{compatibility}(C_\mathcal{S}, C_\mathcal{E}) = \frac{|C_\mathcal{S} \cap C_\mathcal{E}|}{|C_\mathcal{S} \cup C_\mathcal{E}|}$. If the compatibility is bigger than a threshold, we further check whether at least a document in event $\mathcal{E}$ and at least a document in story tree $\mathcal{S}$ share n or more common words in their titles (with stop words removed). If yes, we assign event $\mathcal{E}$ to story tree $\mathcal{S}$. Otherwise, they are not related. In our experiments, we set $n = 1$. If the event $\mathcal{E}$ is not related to any existing story tree, a new story tree will be created.

b) Updating the related story tree. After a related story tree $\mathcal{S}$ has been identified for the incoming event $\mathcal{E}$, we perform one of the 3 types of operations to place event $\mathcal{E}$ in the tree: *merge*, *extend* or *insert*, as shown in Fig. 5. The *merge* operation merges the new event $\mathcal{E}$ into an existing event node in the tree. The *extend* operation will append the event $\mathcal{E}$ as a child node to an existing event node in the tree. Finally, the *insert* operation directly appends event $\mathcal{E}$ to the root node of story tree $\mathcal{S}$. Our system chooses the most appropriate operation to process the incoming event based on the following procedures.

Merge: we merge $\mathcal{E}$ with an existing event in the tree, if they essentially talk about the same event. This can be achieved by checking whether the centroid documents of the two events are

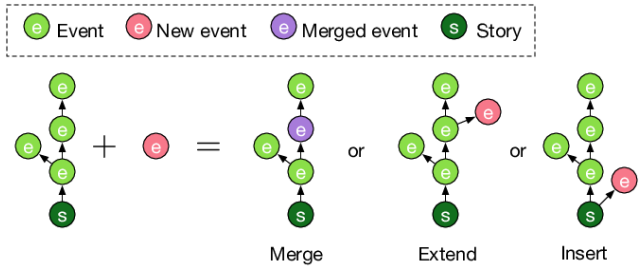


Fig. 5. Three types of operations to place a new event into its related story tree.

talking about the same thing using the document-pair relationship classifier described in Sec. 3.2. The centroid document of an event is simply the concatenation of all the documents in the event. If the centroid documents of the two events are talking about the same thing, we merge the new incoming event with the existing event node. Otherwise, we continue the procedures below.

To merge a new incoming event with an existing event node, we add the set of documents contained in the incoming event to the set of documents belonging to the existing event node. Besides, the set of keywords of the incoming event is also merged into that of the existing event node.

Extend *and* ***Insert***: if event $\mathcal{E}$ does not overlap with any existing event, we will find the parent event node in $\mathcal{S}$ to which it should be appended. We calculate the *connection strength* between the new event $\mathcal{E}$ and each existing event $\mathcal{E}_j \in \mathcal{S}$ based on three factors: 1) the time distance between $\mathcal{E}$ and $\mathcal{E}_j$, 2) the compatibility of the two events, and 3) the *storyline coherence* if $\mathcal{E}$ is appended to $\mathcal{E}_j$ in the tree, i.e.,

$$\begin{aligned}\text{ConnectionStrength}(\mathcal{E}_j, \mathcal{E}) := \ &\text{compatibility}(\mathcal{E}_j, \mathcal{E}) \times \\ &\text{coherence}(\mathcal{L}_{\mathcal{S}\to\mathcal{E}_j\to\mathcal{E}}) \times \text{timePenalty}(\mathcal{E}_j, \mathcal{E}).\end{aligned} \tag{2}$$

Now we explain the three components in the above equation one by one. *First*, the compatibility between two events $\mathcal{E}_i$ and $\mathcal{E}_j$ is given by

$$\text{compatibility}(\mathcal{E}_i, \mathcal{E}_j) = \frac{\text{TF}(d_{c_i}) \cdot \text{TF}(d_{c_j})}{\|\text{TF}(d_{c_i})\| \cdot \|\text{TF}(d_{c_j})\|}, \tag{3}$$

where d_{c_i} is the centroid document of event $\mathcal{E}_i$.

Furthermore, the storyline of $\mathcal{E}_j$ is defined as the path in $\mathcal{S}$ starting from the root node of $\mathcal{S}$ ending at $\mathcal{E}_j$ itself, denoted by $\mathcal{L}_{\mathcal{S}\to\mathcal{E}_j}$. Similarly, the storyline of $\mathcal{E}$ appended to $\mathcal{E}_j$ is denoted by $\mathcal{L}_{\mathcal{S}\to\mathcal{E}_j\to\mathcal{E}}$. For a storyline $\mathcal{L}$ represented by a path $\mathcal{E}^0 \to \ldots \to \mathcal{E}^{|\mathcal{L}|}$, where $\mathcal{E}^0 := \mathcal{S}$, its *coherence* [39] measures the theme consistency along the storyline, and is defined as

$$\text{coherence}(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{i=0}^{|\mathcal{L}|-1} \text{compatibility}(\mathcal{E}^i, \mathcal{E}^{i+1}), \tag{4}$$

Finally, the bigger the time gap between two events, the less possible that the two events are connected. We thus calculate time penalty by

$$\text{timePenalty}(\mathcal{E}_j, \mathcal{E}) = \begin{cases} e^{\delta \cdot (t_{\mathcal{E}_j} - t_{\mathcal{E}})} & \text{if } t_{\mathcal{E}_j} - t_{\mathcal{E}} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

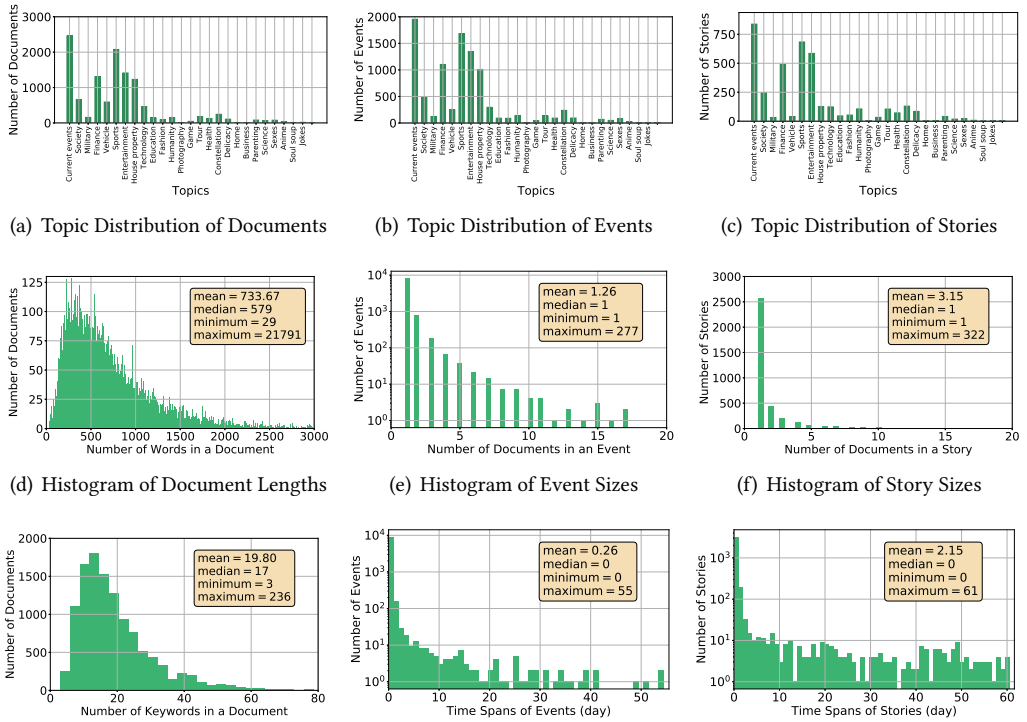
where $t_{\mathcal{E}_j}$ and $t_{\mathcal{E}}$ are the timestamps of event $\mathcal{E}_j$ and $\mathcal{E}$ respectively. The timestamp of an event is the minimum timestamp of all the documents in the event.

We calculate the connection strength between the new event $\mathcal{E}$ and every event node $\mathcal{E}_j \in \mathcal{S}$ using (2), and append event $\mathcal{E}$ to the existing $\mathcal{E}_j$ that leads to the maximum connection strength. If the maximum connection strength is lower than a threshold value, we *insert* $\mathcal{E}$ into story tree $\mathcal{S}$ by directly appending it to the root node of $\mathcal{S}$. In other words, *insert* is a special case of *extend*.

4 PERFORMANCE EVALUATION

We conduct two groups of experiments. The first group independently evaluates *EventX*, since it is a core novelty within our Story Forest system. We report the performance of *EventX* on a manually labeled Chinese News Events dataset and the openly available 20 Newsgroups dataset. We then compare *EventX* against several clustering baselines. In the second group, we add the story visualization structures and examine the overall performance of Story Forest on a much larger news documents dataset, i.e., the Chinese News Corpus dataset. We also investigate the effects of various hyper-parameters in *EventX* as well as the algorithm complexity of the entire system.

4.1 News Datasets



(a) Topic Distribution of Documents (b) Topic Distribution of Events (c) Topic Distribution of Stories

(d) Histogram of Document Lengths (e) Histogram of Event Sizes (f) Histogram of Story Sizes

(g) Histogram of Document Keyword Numbers (h) Histogram of Event Time Spans by Day (i) Histogram of Story Time Spans by Day

Fig. 6. The characteristics of the introduced Chinese News Event and Story dataset.

To help evaluate Story Forest, we create the *Chinese News Corpus* dataset by collecting 60 GB of Chinese news documents (approximately 1.5 millions of news articles) from major Internet news providers in China, such as Tencent and Sina, in a three-month period from October 1, 2016 to December 31, 2016 covering different topics in the open domain. The entire Story Forest system is

tested on the full 60 GB of Chinese news documents and we employ human evaluators to analyze its overall performance.

We then sample a smaller subset from these documents to create the *Chinese News Events* dataset, which is manually labeled with the truth events and stories, by editors and product managers at Tencent. It is also, to the best of our knowledge, the first Chinese dataset for event extraction evaluation. In the creation of *Chinese News Events* dataset, we assume that 1) each article only has one event ID, i.e., associated with the breaking news that leads to this article; 2) different events from the same story are clearly distinguishable, partly because they usually follow the progressing timeline of real-world affairs; 3) each article only belongs to one news story indicated by a unique story ID (we ignore the complex storylines and entangled plots such as those in a fiction). We train and evaluate the *EventX* algorithm on this labeled dataset of 11,748 Chinese news articles. When evaluating the whole Story Forest system on the Chinese News Corpus dataset, documents belonging to the Chinese News Events dataset are removed.

Fig. 6 shows some statistics of the new dataset. Fig. 6(a), Fig. 6(b) and Fig. 6(c) demonstrate how many documents, events and stories each topic contains. We can see that the distributions are highly skewed. The top 3 largest topics are "entertainment", "sports" and "current events". Fig. 6(d) shows the histogram of word counts in documents. The average number of words in a document is 733.67 and the longest document has 21,791 words. From Fig. 6(g), we can see that the average number of keywords extracted from each document is 19.8. In terms of the sizes of events and stories, the histograms of the number of documents in each event and that in each story are shown in Fig. 6(e) and Fig. 6(f), respectively. We can easily observe that for real-world news data, a majority of events only contain a single document: the average size of an event is 1.26 (documents), while the largest event contains 277 documents. The average size of a story is 3.15 (documents) and the largest story contains 322 documents. Finally, Fig. 6(h) and Fig. 6(i) plot the distributions of the time spans of each event and story. The average time span of an event is 0.26 days, and that of a story is 2.15 days. These statistics help to justify the assumptions we made above when labeling the dataset—most breaking news events and stories have relatively short lifespans with simple narratives (in contrast to fictions or scientific articles).

4.2 Evaluation of EventX

4.2.1 Evaluation Datasets. We utilize the following datasets for evaluating *EventX*:

- **Chinese News Events (CNE)**. This is the full dataset that contains 11,748 news articles belonging to 9,327 events. The average number of words in documents is 733.67.
- **Chinese News Events Subset 1**. It is created by removing events containing less than 2 documents in the Chinese News Events dataset. It contains 3,557 news articles belonging to 1,136 events. The average number of words in documents is 768.07.
- **Chinese News Events Subset 2**. We further filter events that contain less than 4 documents in the Chinese News Events dataset. This dataset contains 1,456 news articles belonging to 176 events. The average length of documents is 760.04 words.
- **20 Newsgroups**[3]. This is a classic dataset for the evaluation of text clustering algorithms. It consists of 18,821 documents that belong to 20 different news groups. The average length of documents is 137.85 words.

For Chinese datasets, after word segmentation, we extract keywords from each document using the supervised approach mentioned in Sec. 3.1. The binary classifier to classify whether a word is a keyword to a document is trained on a dataset that we manually labeled. It contains the keywords of 10,000+ documents, including 20,000+ positive keyword samples and 350,000+ negative samples.

[3] http://qwone.com/~jason/20Newsgroups/

On another Chinese keyword extraction evaluation dataset, our classifier achieved a precision of 0.83 and a recall of 0.76, while they are 0.72 and 0.76 respectively if we exclude the GBDT component. The average number of keywords for each document in our proposed Chinese News Events dataset is 19.8. For English documents, we extracted keywords using RAKE [29].

4.2.2 Evaluation Metrics. The evaluation metrics we used include Homogeneity (H), Completeness (C), V-measure score (V) [30] and Normalized Mutual Information (NMI). Homogeneity and completeness are intuitive evaluation metrics based on conditional entropy analysis. Homogeneity is larger if each cluster contains only members from a single class. The completeness is maximized if all members of a ground truth class are assigned to the same cluster. Homogeneity and completeness scores are defined as:

$$H = 1 - \frac{\sum_{c,k} n_{c,k} \log \frac{n_{c,k}}{n_k}}{\sum_c n_c \log \frac{n_c}{N}}, \tag{6}$$

$$C = 1 - \frac{\sum_{c,k} n_{c,k} \log \frac{n_{c,k}}{n_c}}{\sum_k n_k \log \frac{n_k}{N}}, \tag{7}$$

where n_c is the number of documents in class c, and n_k is the number of documents in cluster k. $n_{c,k}$ is the number of documents in class c as well as in cluster k, and N is the number of total documents in the dataset. The V-measure is the harmonic mean between homogeneity and completeness:

$$\mathrm{V} = \frac{2 \times \mathrm{H} \times \mathrm{C}}{\mathrm{H} + \mathrm{C}}. \tag{8}$$

Normalized mutual information [12] is also widely used for text clustering evaluation. It measures the amount of statistical information shared by the ground truth cluster labels and the cluster assignment results. The NMI is 1 if the cluster results perfectly match the truth labels, and it is close to 0 when the cluster labels are randomly generated. Normalized mutual information is formally defined as:

$$NMI = \frac{\sum_{c,k} n_{c,k} \log \frac{n_{c,k} \cdot N}{n_c \cdot n_k}}{\sqrt{(\sum_c n_c \log \frac{n_c}{N})(\sum_k n_k \log \frac{n_k}{N})}} \tag{9}$$

4.2.3 Methods for Comparison. We compare *EventX* to other document clustering methods, as well as methods that utilize both word and document clusters:

- **KeyGraph**: the KeyGraph algorithm [32] clusters documents based on a keyword co-occurrence graph, without the second-layer clustering based on document relationship graphs and document pair relationship classifier.
- **Co-clustering**: this algorithm takes in a term-document matrix and simultaneously maps rows to row-clusters and columns to column-clusters. Then, it calculates the mutual information difference between the current row and column clusters and iteratively updates both clusters until this difference is below a threshold [9].
- **Word-to-Doc Clustering**: this algorithm takes in a joint probability distribution, typically a term-document matrix. It first computes word-clusters to capture the most information about the documents. Relying on this new cluster-document matrix, the algorithm recursively merges the documents into bigger clusters, such that the mutual information loss between document clusters and word clusters is minimized [35].
- **Non-negative Matrix Factorization (NMF)**: this algorithm coverts a corpus into a term-document matrix where each element in the matrix is the TF-IDF value of a word in a document. The factorized basis vectors correspond to different clusters. By examining the

Table 3. Comparing different algorithms on Chinese News Events (CNE) dataset.

Algorithm	CNE (K = 2331)				CNE (K = 4663)				CNE (K = 9327)			
	H	C	V	NMI	H	C	V	NMI	H	C	V	NMI
EventX	**0.990**	0.954	**0.972**	**0.972**	**0.990**	0.954	**0.972**	**0.972**	**0.990**	0.954	**0.972**	**0.972**
KeyGraph	0.811	**0.959**	0.879	0.882	0.811	**0.959**	0.879	0.882	0.811	0.959	0.879	0.882
Co-Clustering	0.962	0.773	0.857	0.863	0.960	0.839	0.896	0.898	0.958	0.901	0.929	0.929
Word-Doc	0.965	0.820	0.887	0.890	0.964	0.891	0.926	0.927	0.959	**0.974**	0.966	0.966
NMF	0.514	0.954	0.668	0.700	0.403	0.940	0.564	0.615	0.231	0.910	0.368	0.458
K-means+LDA	0.266	0.105	0.151	0.167	0.355	0.126	0.186	0.211	0.466	0.147	0.224	0.262
K-means+TF-IDF	0.445	0.173	0.250	0.278	0.470	0.166	0.245	0.279	0.485	0.154	0.234	0.273

Table 4. Comparing different algorithms on Chinese News Events Subset 1.

Algorithm	CNE Subset 1 (K = 568)				CNE Subset 1 (K = 1136)				CNE Subset 1 (K = 2272)			
	H	C	V	NMI	H	C	V	NMI	H	C	V	NMI
EventX	**0.973**	0.841	**0.902**	**0.905**	0.973	0.841	**0.902**	**0.905**	0.973	0.841	0.902	0.905
KeyGraph	0.837	0.846	0.842	0.842	0.837	0.846	0.842	0.842	0.837	0.846	0.842	0.842
Co-Clustering	0.868	0.777	0.820	0.821	0.860	0.850	0.855	0.855	0.847	0.911	0.878	0.879
Word-to-Doc	0.897	0.822	0.858	0.859	0.876	0.914	0.895	0.895	0.850	**0.979**	**0.910**	**0.912**
NMF	0.491	**0.913**	0.638	0.669	0.351	**0.931**	0.510	0.572	0.156	0.714	0.256	0.334
K-means+LDA	0.886	0.351	0.503	0.558	0.956	0.340	0.502	0.570	0.989	0.314	0.477	0.558
K-means+TF-IDF	0.928	0.363	0.522	0.580	**0.975**	0.338	0.502	0.574	**0.991**	0.313	0.476	0.557

Table 5. Comparing different algorithms on Chinese News Events Subset 2.

Algorithm	CNE Subset 2 (K = 88)				CNE Subset 2 (K = 176)				CNE Subset 2 (K = 352)			
	H	C	V	NMI	H	C	V	NMI	H	C	V	NMI
EventX	**0.964**	0.680	0.798	0.810	**0.964**	0.680	0.798	0.810	**0.964**	0.680	0.798	0.810
KeyGraph	0.691	0.767	0.717	0.728	0.691	0.767	0.717	0.728	0.691	0.767	0.717	0.728
Co-Clustering	0.783	0.728	0.755	0.755	0.750	0.801	0.775	0.775	0.723	0.867	0.788	0.792
Word-to-Doc	0.886	0.765	**0.821**	**0.823**	0.805	**0.865**	**0.834**	**0.835**	0.731	**0.927**	**0.817**	**0.823**
NMF	0.753	**0.853**	0.800	0.802	0.801	0.809	0.805	0.805	0.745	0.802	0.773	0.773
K-means+LDA	0.783	0.430	0.555	0.581	0.890	0.408	0.559	0.602	0.940	0.380	0.541	0.598
K-means+TF-IDF	0.857	0.469	0.606	0.634	0.928	0.424	0.582	0.627	0.959	0.379	0.543	0.603

Table 6. Comparing different algorithms on 20 Newsgroups dataset.

Algorithm	20 Newsgroups (K = 10)				20 Newsgroups (K = 20)				20 Newsgroups (K = 40)			
	H	C	V	NMI	H	C	V	NMI	H	C	V	NMI
EventX	**0.996**	0.340	**0.507**	**0.582**	**0.996**	0.340	**0.507**	**0.582**	**0.996**	0.340	**0.507**	**0.582**
KeyGraph	0.995	0.340	0.507	0.581	0.995	0.340	0.507	0.581	0.995	0.340	0.507	0.581
Co-Clustering	0.421	0.318	0.362	0.366	0.374	0.368	0.371	0.371	0.351	0.423	0.384	0.386
Word-to-Doc	0.578	0.306	0.400	0.421	0.540	0.390	0.453	0.459	0.515	**0.464**	0.488	0.489
NMF	0.315	**0.529**	0.395	0.408	0.396	**0.468**	0.429	0.431	0.456	0.413	0.433	0.434
K-means+LDA	0.245	0.352	0.368	0.293	0.333	0.352	0.342	0.342	0.321	0.279	0.298	0.299
K-means+TF-IDF	0.350	0.282	0.312	0.194	0.368	0.264	0.308	0.159	0.332	0.301	0.316	0.236

largest component of a document along any of the vectors, we can decide the cluster membership of that document [40].

- **K-means with LDA**: extract the 1000-dimensional LDA vector of each document, and cluster them by the K-means algorithm [18], which is probably the most widely used method for clustering.
- **K-means with TF-IDF**: represent each document by TF-IDF vector, and cluster by K-means.

For the compared baseline methods, we vary the number of clusters for each dataset. For algorithms that require iterative updates, we set a maximum iteration of 20 (which proved to be enough for convergence). Notice that our *EventX* algorithm and the KeyGraph algorithm do not require a pre-defined number of clusters.

The *EventX* algorithm needs to pre-train a document relationship classifier. For the Chinese News Events dataset and subsets, we trained a document-pair relationship classifier using 5,000 labeled news pairs, collected from the web from January 01, 2017 to April 04, 2017. For the 20 Newsgroups dataset, we used the training set[4] to construct a document-pair relationship dataset and trained a classifier, then evaluated the algorithms on the test set.

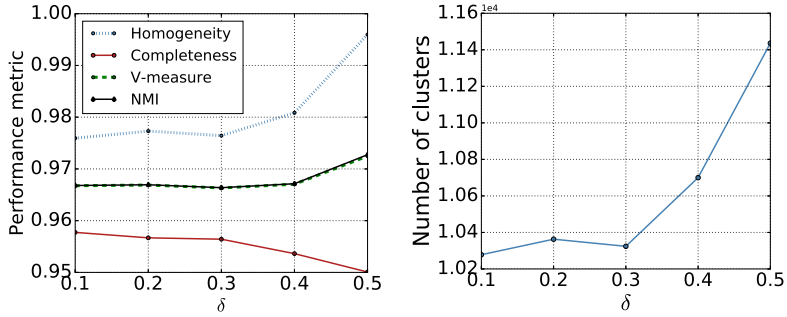
4.2.4 Comparison with Existing Methods. Table 3 shows the performance of different algorithms on the four datasets with different number of clusters K. As we can see, our approach achieves the best V-measure and NMI scores in most cases. And our method achieves overall high homogeneity scores among all datasets. This implies that most event clusters we obtain are pure: each event only contains documents that talk about the same event. In comparison, the homogeneity for other methods varies widely across datasets. The reason is that we adopted two layers of graph-based clustering algorithms to extract events at a more appropriate granularity. For completeness, even though our approach scored a little bit lower than KeyGraph, it is more or less expected. As we further split the topic clusters in the second layer document relationship graph, it is more likely for documents labeled as one class to appear in different clusters, simply because with a two-layered clustering scheme, we have more clusters. Considering the significant improvements of homogeneity, we believe the slightly lower completeness score would not strongly affect the real-world performance of our model.

The performance of the Word-to-Doc Clustering algorithm is slightly better than our algorithm on the Chinese News Events Subset 2. However, our algorithm does not need the number of clusters ahead of time, while the performance of the Word-to-Doc Clustering algorithm is highly dependent on this parameter. For real-world news data, the event size distribution is highly skewed like the Chinese News Events full dataset, rather than the subsets.

Considering the results on the 20 Newsgroups dataset. Our algorithm achieves better performance than the other algorithms in terms of the Homogeneity, V-measure score and NMI. We can also observe that the completeness of our algorithm is relatively low on the 20 Newsgroups dataset, while the homogeneity is still high. The reason is that the documents in 20 Newsgroups dataset are grouped according to topics rather than events. Therefore, two documents belonging to the same cluster may have entirely different keywords, and our algorithm will assign them two different cluster labels. For the same reason, we can see that the performance of KeyGraph is similar to *EventX* on the 20 Newsgroups dataset. After the first layer keyword co-occurrence graph based clustering, the clusters are already pure and the second layer document relationship graph based clustering would not further split it.

4.2.5 Influence of Parameters. The main parameter that will influence the final clustering result of *EventX* is the threshold δ that assigns each document to a keyword subgraph in the first layer clustering step. Fig. 7 shows its influence on the clustering performance on the Chinese News Events dataset. As we can see, the performance is quite robust when $\delta \leq 0.3$. If we keep increasing δ, the number of clusters increases, leading to an increase in homogeneity and a decrease in completeness. The reason is that, when δ is within a reasonable range (usually between 0.1 to 0.3), the performance of our algorithm is insensitive to it, as the first layer keyword co-occurrence graph based clustering will split documents into topics, and the value of δ will only influence the granularity of the clustered topics. The events within each topic will not be divided, and they will be extracted by the second layer clustering procedure. If δ keeps increasing, at some point, events will split in the first layer, resulting in an increase in homogeneity and a decrease in completeness.

[4]The splitting of training and testing set for the 20 Newsgroups is the same with http://csmining.org/index.php/id-20-newsgroups.html.



(a) Clustering performance (b) Number of clusters

Fig. 7. The influence of parameter δ to the clustering performance and number of clusters on the Chinese News Events dataset.

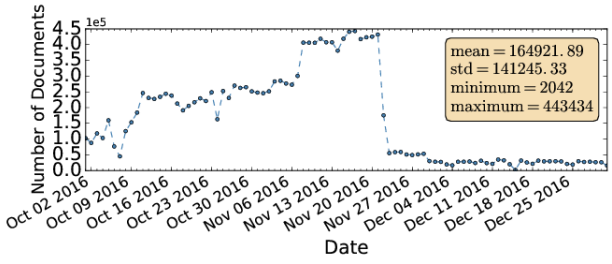


Fig. 8. The number of documents on different days in the Story Forest evaluation dataset.

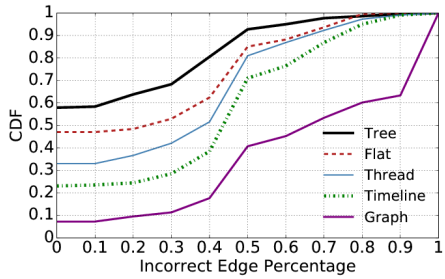
For other parameters, such as the δ_e, δ_p and δ_g we defined in Sec. 3.2.1 and Sec. 3.2.2, they are parameters that influence the construction of keyword co-occurrence graph and subgraphs. Our event extraction algorithm is also insensitive to these parameters as long as they are within a reasonable range. The reason is the same with δ.

4.3 Evaluation of Story Forest

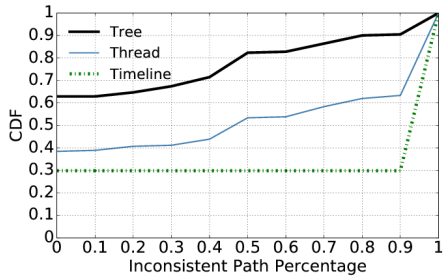
Now we evaluate the complete Story Forest system. Fig. 8 shows the amounts of documents on different days in the larger 60 GB dataset. The average number of documents in one day during that period is 164, 922. For the following experiments, we use the data in the first 7 days for parameter tuning. The remaining data serves as the test set.

We test different event timeline and story generation algorithms on the large 3-month news dataset through pilot user evaluation. To make fair comparisons, the same preprocessing and event extraction procedures before developing story structures are adopted for all methods, with 261 stories detected from the dataset. The only difference is how to construct the story structure given a set of event nodes. We compare our online Story Forest system with the following existing algorithms:

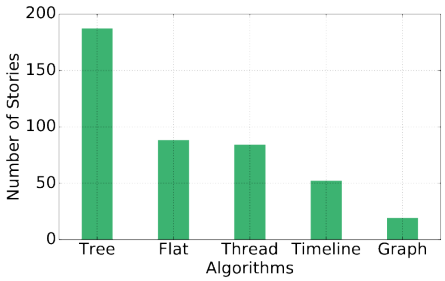
- **Flat Cluster (Flat):** this method clusters related events into a story without revealing the relationships between events, which approximates some previous works in TDT [3, 43].
- **Story Timeline (Timeline):** this method organizes events linearly according to the timestamps of events [31, 32].
- **Story Graph (Graph):** this method calculates a connection strength for every pair of events and connect the pair if the score exceeds a threshold [42].



(a) Percentage of incorrect edges



(b) Percentage of inconsistent paths



(c) Number of times rated as the most readable structure

Fig. 9. Comparing the performance of different story structure generation algorithms.

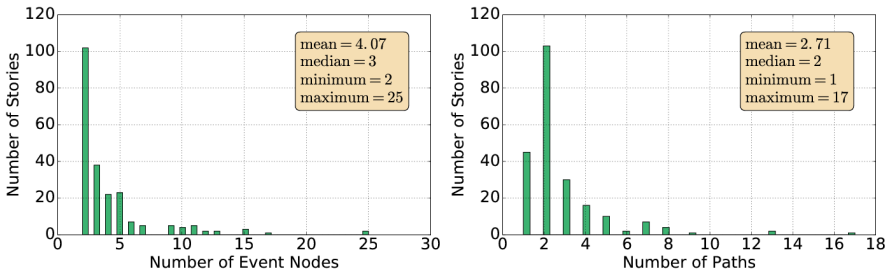
Table 7. Comparing different story structure generation algorithms.

	Tree	Flat	Thread	Timeline	Graph
Correct edges	**82.8%**	73.7%	66.8%	58.3%	32.9%
Consistent paths	**77.4%**	–	50.1%	29.9%	–
Best structure	**187**	88	84	52	19

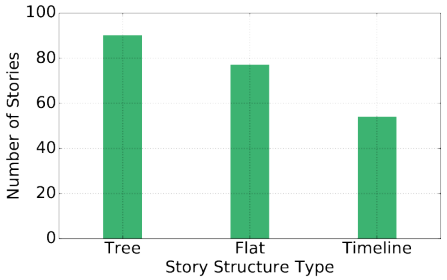
- **Event Threading (Thread):** this algorithm appends each new event to its most similar earlier event [24]. The similarity between two events is measured by the TF-IDF cosine similarity of the event centroids.

We enlisted 10 human reviewers, including product managers, software engineers and senior undergraduate students, to blindly evaluate the results given by different approaches. Each individual story was reviewed by 3 different reviewers. When the reviewers' opinions are different, they will discuss to give a final result. For each story, the reviewers answered the following questions for each of the 5 different structures generated by different schemes:

(1) Do all the documents in each story cluster truly talk about the same story (*yes* or *no*)? Continue if *yes*.
(2) Do all the documents in each event node truly talk about the same event (*yes* or *no*)? Continue if *yes*.
(3) For each story structure given by different algorithms, how many edges correctly represent the event connections?



(a) Histogram of the number of events in each story (b) Histogram of the number of paths in each story



(c) Numbers of different story structures

Fig. 10. The characteristics of the story structures generated by the Story Forest system.

(4) For each story structure given by story forest, event threading and story timeline, how many paths from ROOT to any leaf node exist in the graph? And how many such paths are logically coherent?
(5) Which algorithm generates the structure that is the best in terms of revealing the story's underlying logical structure?

Note that for question (3), the total number of edges for each tree equals to the number of events in that tree. Therefore, to make a fair comparison, for the story graph algorithm, we only retain the n edges with the top scores, where n is the number of events in that story graph.

We first report the clustering effectiveness of our system in the pilot user evaluation on the 3-month dataset. Among the 261 stories, 234 of them are pure story clusters (*yes* to question 1), and furthermore there are 221 stories only contains pure event nodes (*yes* to question 2). Therefore, the final accuracy to extract events (*yes* to both question 1 and 2) is 84.7%.

Next, we compare the output story structures given by different algorithms from three aspects: the correct edges between events, the logical coherence of paths, and the overall readability of different story structures. Fig. 9(a) compares the CDFs of incorrect edge percentage under different algorithms. As we can see, Story Forest significantly outperforms the other 4 baseline approaches. As shown in Table 7, for 58% story trees, all the edges in each tree are reviewed as correct, and the average percentage of correct edges for all the story trees is 82.8%. In contrast, the average correct edge percentage given by the story graph algorithm is 32.9%.

An interesting observation is that the average percentage of correct edges given by the simple flat structure is 73.7%, which is a special case of our tree structures. This can be explained by the fact that most real-world breaking news that last for a constrained time period are not as complicated as a novel with rich logical structure, and a flat structure is often enough to depict their underlying

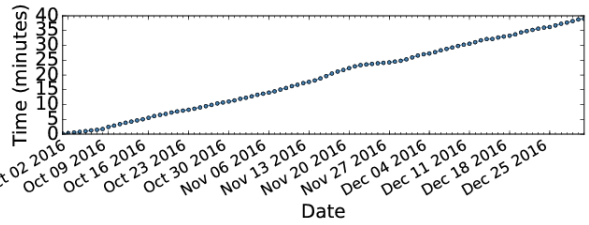


Fig. 11. The running time of our system on the 3-month news dataset.

logic. However, for stories with richer structures and a relatively longer timeline, Story Forest gives better results than other algorithms by comprehensively considering the event similarity, path coherence and time gap, while other algorithms only consider parts of all the factors.

For path coherence, Fig. 9(b) shows the CDFs of percentages of inconsistent paths under different algorithms. Story Forest gives significantly more coherent paths: the average percentage of coherent paths is 77.4% for our algorithm, and is 50.1% and 29.9%, respectively, for event threading and story timeline. Note that path coherence is meaningless for flat or graph structure.

Fig. 9(c) plots the overall readability of different story structures. Among the 221 stories, the tree-based Story Forest system gives the best readability on 187 stories, which is much better than all other approaches. Different algorithms can generate the same structure. For example, the Story Forest system can also generate a flat structure, a timeline, or the same structure as the event threading algorithm does. Therefore, the sum of the number of best results given by different approaches is bigger than 221. It's worth noting that the flat and timeline algorithms also give 88 and 52 most readable results, which again indicates that the logic structures of a large portion of real-world news stories can be characterized by simple flat or timeline structures, which are special cases of story trees. And complex graphs are often an overkill.

We further inspect the story structures generated by Story Forest. Fig. 10(a) and Fig. 10(b) plot the distributions of the number of events and the number of paths in each story tree, respectively. The average numbers of events and paths are 4.07 and 2.71, respectively. Although the tree structure includes the flat and timeline structures as special cases, among the 221 stories, Story Forest generates 77 flat structures and 54 timelines, while the remaining 90 structures generated are still story trees. This implies that Story Forest is versatile and can generate diverse structures for real-world news stories, depending on the logical complexity of each story.

4.4 Algorithm Complexity and System Overhead

In this section, we discuss the complexity of each step in the Story Forest. For a time slot (e.g., in our case is one day), let N_d be the number of documents, N_w the number of unique words in corpora, note $N_w << N_d$, N_e the number of different events, N_s the number of different stories, and N_k represents the maximum number of unique keywords in a document.

As discussed in [32], building keyword graph requires $O(N_d N_k + N_w^2)$ complexity, and community detection based on betweenness centrality requires $O(N_w^3)$. The complexity of assigning documents to keyword communities is $O(N_d N_k N_e)$. So by far the total complexity is $O(N_d N_k N_e + N_w^3)$. There exist other community detection algorithms requiring only $O(N_w^2)$, such as the algorithm in [27]. Thus we can further improve efficiency by using faster community detection algorithms.

After clustering documents by keyword communities, for each cluster the average number of documents is N_d/N_e. The pair-wise document relation classification is implemented in $O((N_d/N_e)^2)$. The complexity of the next document graph splitting operation is $O((N_w/N_e)^3)$. Therefore, the total complexity is $O(N_e((N_d/N_e)^2 + (N_w/N_e)^3))$. Our experiments show that usually $1 \leq N_d/N_e \leq 100$. Combining with $N_w << N_d$, the complexity is now approximately $O(N_e)$.

To grow story trees with new events, the complexity of finding the related story tree for each event is of $O(N_s T)$, where T is the history length to keep existing stories and delete older stories. If no existing related story, creating a new story requires $O(1)$ operations. Otherwise, the complexity of updating a story tree is $O(T^{N_e}/_{N_s})$. In summary, the complexity of growing story trees is $O(N_e T(N_s + {}^{N_e}/_{N_s})) \approx O(TN_eN_s)$, as our experience on the Tencent news dataset shows that $1 \leq {}^{N_e}/_{N_s} \leq 200$. Our online algorithm to update story structure requires $O({}^{N_e}/_{N_s})$ complexity and delivers a consistent story development structure, while most existing offline optimization based story structure algorithms require at least $O(({}^{N_e}/_{N_s})^2)$ complexity and disrupt the previously generated story structures.

Fig. 11 shows the running time of our *Story Forest* system on the 3 months news dataset. The average time of processing each day's news is around 26 seconds, and increases linearly with the number of days. For the offline keyword extraction module, the processing efficiency is approximately 50 documents per second. The performance of the keyword extraction module is consistent with time and doesn't require frequently retraining. The LDA model is incrementally retrained every day to handle new words. For keyword extraction, the efficiency of event clustering and story structure generation can be further improved by a parallel implementation.

5 RELATED WORK

There are mainly two research lines that are highly related to our work: Text Clustering and Story Structure Generation.

5.1 Text Clustering

The problem of text clustering has been well studied by researchers [1]. Distance based clustering algorithms measure the closeness between text pieces with similarity functions such as cosine similarity. Various representations, such as TF-IDF, BM25 term weighting [7], can be utilized to represent a text object. After transforming text into features, different strategies can be applied for clustering. Partition-based algorithms such as K-means [18] or K-medoids [26] divide the corpus into pre-defined number of clusters. The Spherical K-means algorithm [6] is especially suitable for text clustering due to its low memory and computational requirement. However, such algorithms are sensitive to variations in parameter values and need to specify the number of clusters. The selection of features also plays a key role in the final performance of clustering [21]. Hierarchical algorithms [13] recursively find nested clusters and create a tree-like structure, but they still need to assume the number of clusters or a similarity threshold. Density-based algorithms [11] do not need to specify the number of clusters in advance, but they do not scale well to high-dimensional sparse data like text [18].

Word and phrase based algorithms find important clusters of words or phrases. [4] clusters documents based on frequent pattern mining. [35] proposes a two-phase clustering procedure that finds word-clusters such that most of the mutual information between words and documents is preserved, and leverages the word-clusters to perform document clustering. Co-clustering algorithms [9] simultaneously cluster words and documents, as the problem of clustering words and clustering documents are closely related. There are also approaches which utilize the document keywords co-occurrence information to construct a keyword graph, and clustering documents by applying community detection techniques on the keyword graph [32].

Non-negative Matrix Factorization is particularly suitable for clustering as a latent space method [40]. It factorizes a term-document matrix, where the vectors in the basis system directly correspond to different clusters. It has been shown that matrix factorization is equivalent to spectral clustering [10].

Probabilistic model-based algorithms aim to create a probabilistic generative model for text documents. Topic models such as Latent Dirichlet Allocation (LDA) [5] and Probabilistic Latent Semantic Indexing (PLSA) [15] assume documents are generated by multiple topics. The Gaussian Mixture Model (GMM) [14] assumes that data points are generated by a mixture of Gaussian distributions. However, such model-based algorithms are computationally intensive and do not produce satisfying results when clustering at a finer granularity.

There are also some works concerning the events described in text objects. [36] presents a news event extraction system to extract violent and disaster events from online news. [28] proposes a system to extract an open-domain calendar of significant events from Twitter. In contrast, our EventX algorithm is specially tailored for event extraction among news documents in the open domain. The length of news articles are relatively long compared to Twitters, and the types of events are not restricted to violent and disaster events.

5.2 Story Structure Generation

The Topic Detection and Tracking (TDT) research spot news events and group by topics, and track previously spotted news events by attaching related new events into the same cluster [2, 3, 32, 42]. However, the associations between related events are not defined or interpreted by TDT techniques. To help users capture the developing structure of events, different approaches have been proposed. [24] proposed the concept of *Event Threading*, and tried a series of strategies based on similarity measure to capture the dependencies among events. [42] combines the similarity measure between events, temporal sequence and distance between events, and document distribution along the timeline to score the relationship between events, and models the event evolution structure by a directed acyclic graph (DAG). [22] discover and summarize the evolutionary patterns of themes in a text stream by first generating word clusters for each time period and then use the Kullback-Leibler divergence measure to discover coherent themes over time.

The above research works measure and model the relationship between events in a pairwise manner. However, the overall story consistency is not considered. [37] generates story summarization from text and image data by constructing a multi-view graph and solving a dominating set problem, but it omits the consistency of each storyline. The *Metro Map* model proposed in [34] defines metrics such as coherence and diversity for story quality evaluation, and identifies lines of documents by solving an optimization problem to maximize the topic diversity of storylines while guaranteeing the coherence of each storyline. [39] further summarize documents with key images and sentences, and then extract story lines with different definitions of coherence and diversity. These works consider the problem of discovering story development structure as optimizing problems with given news corpora. However, new documents are being generated all the time, and systems that are able to catch related news and update story structures in an online manner are desired.

As studies based on unsupervised clustering techniques [41] perform poorly in distinguishing storylines with overlapped events [16], more recent works introduce different Bayesian models to generate storyline. However, they often ignore the intrinsic structure of a story [17] or fail to properly model the hidden relations [44]. [16] proposes a hierarchical Bayesian model for storyline generation, and utilize twitter hashtags to "supervise" the generation process. However, the Gibbs sampling inference of the model is time consuming, and such twitter data is not always available for every news stories.

6 CONCLUDING REMARKS AND FUTURE WORKS

In this paper, we describe our experience of implementing *Story Forest*, a news content organization system at Tencent, which is designed to discover events from vast streams of trending and breaking news and organize events in sensible story trees in an online manner. Our system is specifically

tailored for fast processing massive amounts of breaking news data, whose story structures can most likely be captured by either a tree, a timeline or a flat structure. We proposed *EventX*, a semi-supervised text clustering algorithm designed to efficiently extract events from vast news corpora. *EventX* is a two-layered, graph-based document clustering scheme. In the first layer, we leveraged keyword co-occurrence graphs to cluster documents into topics, while in the second layer, we utilized a novel document relationship graph constructed through supervised learning to further split each topic into events, leading to an overall semi-supervised learning procedure. Our system further organizes the events into story trees with efficient online algorithms upon the arrival of daily news data.

We conducted extensive performance evaluation based on 60 GB of real-world (Chinese) news data, although our ideas are not language-dependent and can easily be extended to other languages, through detailed pilot user experience studies. To stimulate future research in event extraction, we also created a large Chinese News Events dataset that contains 11,748 long news articles collected from all major Internet news portals in China from October 01, 2016 to November 30, 2016, with ground truth event labels. We evaluated *EventX* on the Chinese News Events dataset, as well as the 20 Newsgroups English dataset. Extensive results suggest that our clustering procedure is significantly more effective at accurate event extraction than existing algorithms. For event extraction, the V-measure score and NMI of the clustering results on the Chinese News Events dataset are both 0.972, demonstrating the effectiveness of *EventX* to extract conceptually clean event clusters from vast and unstructured Internet news data. For story generation, 83% of the event links generated by Story Forest are logically correct as compared to an accuracy of 33% generated by more complex story graphs, demonstrating the ability of our system to organize trending news events into a logical structure that appeals to human readers.

In our future work, we will extend our proposed Story Forest and EventX framework to make it applicable to a continuous stream of news documents. Currently, we can use EventX in a batch-based manner: for example, we can extract the events for each batch of documents collected in different hours, and merge a new event with an existing event if they are talking about the same one. We can further improve our framework by maintaining a dynamic keyword graph, performing dynamic keyword community detection, and run incremental event extraction based on such a dynamic keyword graph. Whenever a new document stream comes in, we update the keyword graph to find out new keyword communities, and cluster new documents into new events or assign them to existing events. In this way, our framework can be directly applied to a continuous stream of news documents, rather than processing them in batches.

REFERENCES

[1] Charu C Aggarwal and ChengXiang Zhai. 2012. A survey of text clustering algorithms. In *Mining text data*. Springer, 77–128.

[2] James Allan. 2012. *Topic detection and tracking: event-based information organization*. Vol. 12. Springer Science & Business Media.

[3] James Allan, Ron Papka, and Victor Lavrenko. 1998. On-line new event detection and tracking. In *SIGIR*. ACM, 37–45.

[4] Florian Beil, Martin Ester, and Xiaowei Xu. 2002. Frequent term-based text clustering. In *KDD*. ACM, 436–442.

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *JMLR* 3, Jan (2003), 993–1022.

[6] Christian Buchta, Martin Kober, Ingo Feinerer, and Kurt Hornik. 2012. Spherical k-means clustering. *Journal of Statistical Software* 50, 10 (2012), 1–22.

[7] Stefan Büttcher, Charles LA Clarke, and Brad Lushman. 2006. Term proximity scoring for ad-hoc retrieval on very large text collections. In *SIGIR*. ACM, 621–622.

[8] Pi-Chuan Chang, Michel Galley, and Christopher D Manning. 2008. Optimizing Chinese word segmentation for machine translation performance. In *ACL*. 224–232.

[9] Inderjit S Dhillon, Subramanyam Mallela, and Dharmendra S Modha. 2003. Information-theoretic co-clustering. In *KDD*. ACM, 89–98.

[10] Chris Ding, Xiaofeng He, and Horst D Simon. 2005. On the equivalence of nonnegative matrix factorization and spectral clustering. In *SDM*. SIAM, 606–610.
[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *KDD*, Vol. 96. 226–231.
[12] Pablo A Estévez, Michel Tesmer, Claudio A Perez, and Jacek M Zurada. 2009. Normalized mutual information feature selection. *IEEE TNN* 20, 2 (2009), 189–201.
[13] Benjamin CM Fung, Ke Wang, and Martin Ester. 2003. Hierarchical document clustering using frequent itemsets. In *SDM*. SIAM, 59–70.
[14] Xiaofei He, Deng Cai, Yuanlong Shao, Hujun Bao, and Jiawei Han. 2011. Laplacian regularized gaussian mixture model for data clustering. *IEEE TKDE* 23, 9 (2011), 1406–1418.
[15] Thomas Hofmann. 1999. Probabilistic latent semantic indexing. In *SIGIR*. ACM, 50–57.
[16] Ting Hua, Xuchao Zhang, Wei Wang, Chang-Tien Lu, and Naren Ramakrishnan. 2016. Automatical Storyline Generation with Help from Twitter. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 2383–2388.
[17] Lifu Huang and Lian'en Huang. 2013. Optimized Event Storyline Generation based on Mixture-Event-Aspect Model.. In *EMNLP*. 726–735.
[18] Anil K Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern recognition letters* 31, 8 (2010), 651–666.
[19] Heeyoung Lee, Marta Recasens, Angel Chang, Mihai Surdeanu, and Dan Jurafsky. 2012. Joint entity and event coreference resolution across documents. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 489–500.
[20] Bang Liu, Di Niu, Haojie Wei, Jinghong Lin, Yancheng He, Kunfeng Lai, and Yu Xu. 2019. Matching Article Pairs with Graphical Decomposition and Convolutions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 6284–6294. https://doi.org/10.18653/v1/P19-1632
[21] Luying Liu, Jianchu Kang, Jing Yu, and Zhongliang Wang. 2005. A comparative study on unsupervised feature selection methods for text clustering. In *Natural Language Processing and Knowledge Engineering, 2005. IEEE NLP-KE'05. Proceedings of 2005 IEEE International Conference on*. IEEE, 597–601.
[22] Qiaozhu Mei and ChengXiang Zhai. 2005. Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In *KDD*. ACM, 198–207.
[23] Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into texts. ACL.
[24] Ramesh Nallapati, Ao Feng, Fuchun Peng, and James Allan. 2004. Event threading within news topics. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*. ACM, 446–453.
[25] Yukio Ohsawa, Nels E Benson, and Masahiko Yachida. 1998. KeyGraph: Automatic indexing by co-occurrence graph based on building construction metaphor. In *Research and Technology Advances in Digital Libraries, 1998. ADL 98. Proceedings. IEEE International Forum on*. IEEE, 12–18.
[26] Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert systems with applications* 36, 2 (2009), 3336–3341.
[27] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. 2004. Defining and identifying communities in networks. *PNAS* 101, 9 (2004), 2658–2663.
[28] Alan Ritter, Oren Etzioni, Sam Clark, et al. 2012. Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1104–1112.
[29] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text Mining* (2010), 1–20.
[30] Andrew Rosenberg and Julia Hirschberg. 2007. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure.. In *EMNLP-CoNLL*, Vol. 7. 410–420.
[31] Hassan Sayyadi, Matthew Hurst, and Alexey Maykov. 2009. Event detection and tracking in social streams.. In *Icwsm*.
[32] Hassan Sayyadi and Louiqa Raschid. 2013. A graph analytical approach for topic detection. *ACM TOIT* 13, 2 (2013), 4.
[33] Dafna Shahaf, Carlos Guestrin, and Eric Horvitz. 2012. Trains of thought: Generating information maps. In *Proceedings of the 21st international conference on World Wide Web*. ACM, 899–908.
[34] Dafna Shahaf, Jaewon Yang, Caroline Suen, Jeff Jacobs, Heidi Wang, and Jure Leskovec. 2013. Information cartography: creating zoomable, large-scale maps of information. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1097–1105.
[35] Noam Slonim and Naftali Tishby. 2000. Document clustering using word clusters via the information bottleneck method. In *SIGIR*. ACM, 208–215.
[36] Hristo Tanev, Jakub Piskorski, and Martin Atkinson. 2008. Real-time news event extraction for global crisis monitoring. In *International Conference on Application of Natural Language to Information Systems*. Springer, 207–218.

[37] Dingding Wang, Tao Li, and Mitsunori Ogihara. 2012. Generating Pictorial Storylines Via Minimum-Weight Connected Dominating Set Approximation in Multi-View Graphs.. In *AAAI*. Citeseer.
[38] Lu Wang, Claire Cardie, and Galen Marchetti. 2016. Socially-informed timeline generation for complex events. *arXiv preprint arXiv:1606.05699* (2016).
[39] Shize Xu, Shanshan Wang, and Yan Zhang. 2013. Summarizing Complex Events: a Cross-Modal Solution of Storylines Extraction and Reconstruction.. In *EMNLP*. 1281–1291.
[40] Wei Xu, Xin Liu, and Yihong Gong. 2003. Document clustering based on non-negative matrix factorization. In *SIGIR*. ACM, 267–273.
[41] Rui Yan, Xiaojun Wan, Jahna Otterbacher, Liang Kong, Xiaoming Li, and Yan Zhang. 2011. Evolutionary timeline summarization: a balanced optimization framework via iterative substitution. In *SIGIR*. ACM, 745–754.
[42] Christopher C Yang, Xiaodong Shi, and Chih-Ping Wei. 2009. Discovering event evolution graphs from news corpora. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 39, 4 (2009), 850–863.
[43] Yiming Yang, Jaime Carbonell, Ralf Brown, John Lafferty, Thomas Pierce, and Thomas Ault. 2002. Multi-strategy learning for topic detection and tracking. In *Topic detection and tracking*. Springer, 85–114.
[44] Deyu Zhou, Haiyang Xu, and Yulan He. 2015. An Unsupervised Bayesian Modelling Approach for Storyline Detection on News Articles.. In *EMNLP*. 1943–1948.