



WebKey: a graph-based method for event detection in web news

Elham Rasouli¹ · Sajjad Zarifzadeh¹ · Amir Jahangard Rafsanjani¹

Received: 31 March 2019 / Revised: 9 August 2019 / Accepted: 14 August 2019 /
Published online: 5 September 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract

With rapid and vast publishing of news over the Internet, there is a surge of interest to detect underlying hot events from online news streams. There are two main challenges in event detection: accuracy and scalability. In this paper, we propose a fast and efficient method to detect events in news websites. First, we identify bursty terms which suddenly appear in a lot of news documents. Then, we construct a novel co-occurrence graph between terms in which nodes and edges are weighted based on important features such as click and document frequency within burst intervals. Finally, a weighted community detection algorithm is used to cluster terms and find events. We also propose a couple of techniques to reduce the size of the graph. The results of our evaluations show that the proposed method yields a much higher precision and recall than past methods, such that their harmonic mean is improved by at least 40%. Moreover, it reduces the running time and memory usage by a factor of at least 2.

Keywords Topic detection and tracking · Event detection · Data mining · Community detection

1 Introduction

Nowadays, reading online news is very common among people. Due to the rapid growth of information on the web, it is evident that a person cannot get all interesting events fast and conveniently, since it is a time-consuming task. Moreover, in many companies, decisions are made based on daily events around the world. Thus, there is a need for a system to extract and report important real-world events from huge web information. Initially, Topic

✉ Sajjad Zarifzadeh
szarifzadeh@yazd.ac.ir

Elham Rasouli
rasoulie@stu.yazd.ac.ir

Amir Jahangard Rafsanjani
jahangard@yazd.ac.ir

¹ Computer Engineering Department, Yazd University, Yazd, Iran

Detection and Tracking (TDT) project started with a pilot study in 1996, supported by DARPA (Allan et al. 1998). They used multiple news sources such as newswires and transcribed speech from television and radio news. Generally, three major concepts are introduced in TDT researches (Allan 2002; Yang et al. 2009):

- *Event* is a happening, which occurs at a specific time and place, e.g. Plasco building fire in Tehran, or airplane crash in Shahrekord (a city in Iran).
- *Story* is a news article that delivers information about the occurrence of an event. Each document in an online news website is considered as a story.
- *Topic* is a set of news stories which are strongly relevant to an event.

The goal of event detection is to find the first story discussing an event and then categorizing stories into topics, based on detected events. It is an unsupervised learning problem which can be divided into retrospective (offline) and online categories (Yang et al. 1999; Atefeh and Khreich 2015). The first category refers to the detection of unidentified events in a corpus of documents, usually using hierarchical clustering algorithms such as Hierarchical Agglomerative Clustering (HAC) (Dai and Sun 2010). Such algorithms require the entire document corpus to be able to categorize topics. The second category refers to the real-time detection of events in news feeds. Since there is no prior information about events, this process is recognized as a query-free retrieval task. As opposed to the former category, clustering techniques used in this category are usually based on incremental (greedy) algorithms that process the input sequentially and merge the same events in one cluster (Hu et al. 2017).

There is also a separate line of research on event detection that employs graph modeling to solve the problem (Sayyadi and Raschid 2013; Mori et al. 2006). In this approach, a co-occurrence graph, also known as KeyGraph (Sayyadi and Raschid 2013), is built between terms of documents where there is an edge between two terms if they appear in the same document. Then, various algorithms, including community detection or ranking algorithms, are used to detect events (Sayyadi and Raschid 2013; Ge et al. 2016). Several studies have shown the advantage of this approach over former methods (Sayyadi and Raschid 2013).

From another perspective, event detection algorithms follow either of document-pivot and feature-pivot approaches. The former is usually performed by clustering documents based on the similarity of their contents (Dai and Sun 2010; Yang et al. 1998). In the latter, events are modeled as a bursty activity in which bursty features are extracted using feature distribution analysis (Fung et al. 2005; Zhao et al. 2012). The burst of a term is a sudden rise in its occurrence frequency in arriving documents within a specific time interval, e.g. one hour or one day (Ge et al. 2016; Weng and Lee 2011). Kleinberg proposed two finite and infinite state automata to model the term flow and discover bursts in an interval (Kleinberg 2003). His finite state machine is widely referred to in the literature (Zhao et al. 2012; Xiaomei et al. 2018).

In this paper, we propose a new feature-pivot graph-based method to detect events in news websites. First, we identify bursty terms from streams of news documents using Kleinberg's state machine (Kleinberg 2003). Then, a co-occurrence graph, called WebKey, is constructed between bursty terms. We suggest the idea of weighing this graph to improve the quality of event detection. The weight of nodes and edges in WebKey are assigned using a combination of important features such as click frequency and document frequency which are calculated within the burst interval of terms. Finally, we use a weighted community detection algorithm to detect events. Generally, community detection algorithms have a high time complexity; therefore, we use a couple of graph reduction techniques, such as graph sampling, to decrease the running time. The results of our experiments on a news dataset

collected from more than 100 Persian news websites show that the precision and recall of our method are improved significantly, compared with a couple of past methods. Moreover, its running time and memory usage are reduced by a factor of at least 2. We also add an online extension to our method which detects events much faster than the original one.

This paper is organized as follows: In the next section, we review the related works and in Section 3, we describe our proposed method. The evaluation results are discussed in Section 4. Finally, we conclude the paper in Section 5.

2 Related works

As mentioned before, event detection algorithms are generally divided into two categories: Retrospective (Offline) Event Detection and Online Event Detection. In this section, we discuss them in both news and social media.

2.1 Retrospective (Offline) event detection

Yang et al. introduced a clustering algorithm called Group-Average Clustering (GAC) which is an agglomerative clustering algorithm, proposed for batch processing. They used GAC to organize documents into topic clusters in TDT corpus (1998). Li and Yamanishi proposed a topic model that uses the distribution of words to represent topics. They tested their model on Reuters news articles (2000). Similarly, Wartene and Brussee investigated different distributional similarity measures on Wikipedia articles to detect topics (2008). Fung et al. proposed a probabilistic approach to cluster documents as events in an English newspaper in Hong Kong. Their algorithm models word occurrences as binomial distribution to extract bursty features and detect events (2005). Likewise, Li et al. proposed a probabilistic model which employs both content and time information of news articles (2005). Dai et al. suggested agglomerative algorithms to detect events by incorporating named entities as features. They also proposed a method to calculate the weights of features (2010). Borsje et al. introduced lexico-semantic patterns to extract events from several RSS news feeds. They proposed an ontology framework that makes use of such patterns to detect events more accurately (2010).

Modeling word co-occurrence as a graph of terms has been used in many event detection research works (Mori et al. 2006). Mori et al. proposed an elegant graph-based approach to detect and track topics in news web pages (2006). They first defined the KeyGraph data structure and then employed suffix tree clustering on it to find events. KeyGraph is a graph of keywords (including terms, noun phrases and named entities) and their co-occurrences which are extracted from textual contents of documents. Each node in this graph represents a keyword and each edge indicates the co-occurrence of two end keywords in the same document, i.e. if two keywords co-occur in at least one document, there is an edge between their corresponding nodes in the KeyGraph. Sayyadi et al. also adopted the KeyGraph idea to represent the relationships between document terms. However, they calculated the edge betweenness centrality, originally introduced for community detection algorithms, to detect events (2013, 2009). Betweenness centrality of an edge is the number of shortest paths passing through that edge. They utilized the Breadth First Search (BFS) algorithm (Cormen et al. 2009a) to find shortest paths in the graph. As a result, the edges lying between communities gain a higher betweenness centrality score and are removed from the graph. Each resulting community represents a topic and each keyword in the community is considered as a feature for that topic. Finally, the likelihood of each topic being associated with a document

is determined by the cosine similarity. Sayyadi et al. compared their method with LDA and showed that it leads to a much better performance. One shortcoming of the KeyGraph is that it is unweighted in a sense that it does not pay attention to the frequency of terms and their co-occurrences in the graph.

In 2016, another graph-based method was introduced in Ge et al. (2016) that uses a version of Kleinberg's method (Kleinberg 2003) to build a graph of bursty keywords. The authors also proposed two centroid-aware event detection models; Node-based Detection Model (NDM) and Area-based Detection Model (ADM). In both models, key nodes are detected and then clusters of events are formed around the nodes. In NDM, each cluster is comprised of bursty keywords (or nodes), while in ADM, clusters are made of overlapping documents (or areas). To obtain centroids, they first calculated the PageRank score for every node and then sorted nodes based on their scores. They devised an iterative algorithm in which in each iteration, the node with the highest score that does not belong to any community, is selected as a centroid and then, a community is constructed with the nearest free neighbors of that centroid.

Chen et al. proposed another graph-based method, which works upon the KeyGraph structure (2017). However, they incorporated semantic relations between keywords into the graph using WordNet (Miller 1995). To detect events, they used modularity maximization and pruned the graph using Markov decision process. Another recent study proposed an offline algorithm which generates a semantic graph using different semantic similarity measures, such as relative edit distance of keywords (Wei et al. 2018). Then, an edge betweenness algorithm is used to detect events in the constructed semantic graph.

2.2 Online event detection

Incremental algorithms have been widely used for real-time detection of events in news media (Hu et al. 2017; Yang et al. 1998). Yang et al. made use of a single-pass clustering algorithm to detect events (Hu et al. 2017). In this algorithm, documents are sequentially clustered based on their content similarity. Hu et al. proposed an adaptive online algorithm that utilizes word embedding to reduce dimension (2017). They clustered semantically similar words using K-means algorithm and then performed another clustering using time slicing techniques where documents are clustered based on their publishing time. Finally, a single-pass algorithm is executed on each cluster to detect events.

Since social media is a great source of information, it has attracted a lot of attention in recent years. Many researchers have concentrated on the event detection problem in social networks, especially in Twitter. Twitter is one of the most popular social networks which provides microblogging service. Messages posted on Twitter are called "tweets". Tweets do not always report a real-world event and they may include a lot of noises, e.g. grammatical and spelling errors, informal language, rumors and fake news. Sankaranarayanan et al. proposed TwitterStand that reports breaking news based on geographical information in tweets (2009). It devised a leader-follower clustering algorithm to reduce noises. Cataldi et al. proposed an algorithm that models the life cycle of terms using aging theory (2010). They also made use of users' relationships and the PageRank algorithm to incorporate authority into content. Becker et al. utilized user-provided annotations to detect events in social media sites such as YouTube, Flickr and Facebook (2010). Phuvipadawat and Murata suggested a method to detect and track breaking news in Twitter (2010). They first downloaded tweets that contain pre-defined hashtags (e.g. #breakingnews) using Twitter API. Next, they indexed the collected tweets and grouped similar ones. The similarity between tweets is calculated using TF-IDF, while boosting proper nouns, hashtags and usernames. Finally, they

ranked clusters with a weighted combination of reliability and popularity. Aggarwal and Subbian integrated network structure with the document content to detect events in Twitter (2012).

Zhao et al. used Kleinberg's method (Kleinberg 2003) to extract bursty features (or keywords) (2012). They weighed obtained features based on TF-IDF in the burst interval and then, clustered them using cosine similarity. Their proposed method is called split-cluster-merge, since the dataset is split into small parts, then each part is clustered separately and finally, similar clusters are merged. Sakaki et al. proposed a method to detect specific events such as earthquakes in tweets (2010, 2013) They formulated the event detection as a classification problem. They utilized the SVM classifier with statistical features such as the number of words in each tweet and contextual features such as words related to the user's query.

Instead of using word co-occurrence patterns, Petkos et al. proposed a soft frequent pattern mining algorithm (SFPM) to detect events in Twitter (2014). In contrast, Garg and Kumar suggested a method that analyzes a directed weighted graph to detect events. The graph is build based on word co-occurrence in Twitter. They compared their method against SFPM and reported a lower redundancy rate (2018). Xiaomei et al. utilized sentiment analysis and hashtags to detect bursty events in Sina Weibo, a Chinese social network. They also used Kleinberg's method (2003) for burst detection.

In this paper, we adopt the idea of constructing co-occurrence graph between bursty terms. However, we weigh our graph based on some new features that can help to precisely detect events. Moreover, we improve the running time and memory usage of the proposed method with a number of graph reduction techniques. We also present an online extension to our retrospective method.

3 Architecture of WebKey method

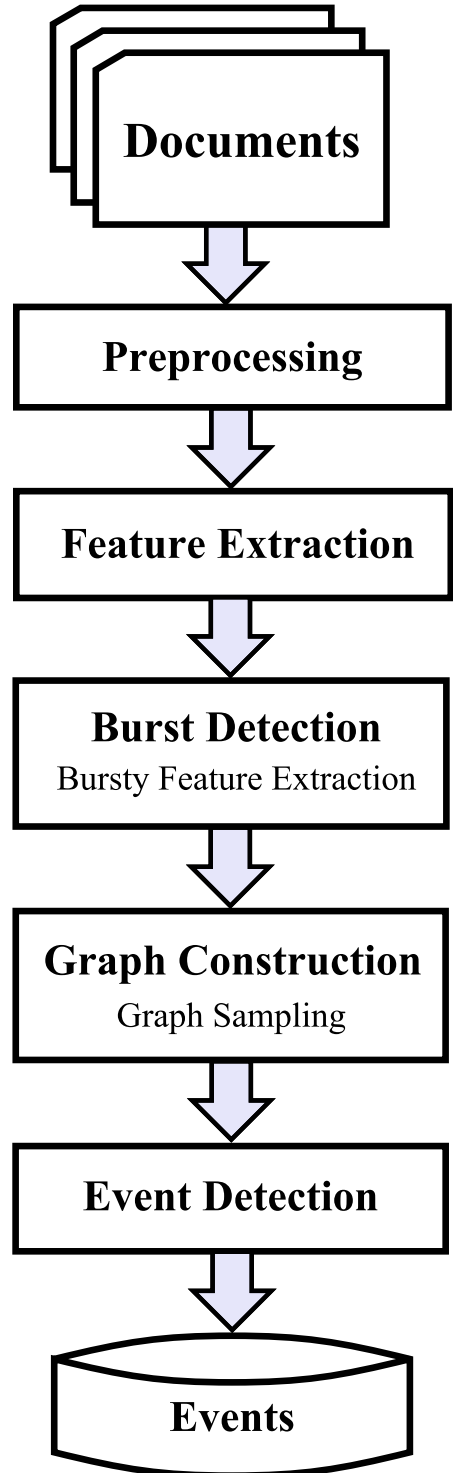
In this section, we propose a new event detection method, called *Weighted Bursty KeyGraph* (*WebKey*), which works based on the retrospective KeyGraph method (Xiaomei et al. 2018). It includes several processing and storing steps which will be discussed in details in this section. Figure 1 demonstrates the overall architecture of WebKey. The input to the system is a collection of news pages or documents crawled from Persian news websites. Each document contains title, summary, content and publishing time (all text fields are in Persian language). We also collect the number of clicks on each document during 1-minute intervals. We discuss more about our dataset in Section 4. We should emphasize that although our dataset is in Persian, but we do not use any language-dependent feature in our method. Therefore, it can be simply applied on other datasets as well.

First, we preprocess our data by removing stop-words, special characters (like %, #, @) and punctuations from the text. We also stem our texts using open-source stemming tools, such as PersianStemmer (Taghi-Zadeh et al. 2017).

3.1 Feature extraction

N -gram is a continuous sequence of N items from the text where items can be letters, words, characters. N represents the number of items in the sequence. For example, bi-gram and tri-gram consists of two and three items in the sequence, respectively. We consider items as words. For instance, in this sentence "There was a heavy rain in Tehran last night", after removing stop words, we have the following four bigrams: "heavy rain", "rain Tehran", "Tehran last" and "last night".

Fig. 1 The architecture of WebKey



We first extract unique unigrams, bigrams, and trigrams from the title and summary of news documents. Instead of using the whole news content, we focus on title and summary fields to reduce the size of our problem and limit the running time, while keeping the precision high. Later, we analyze the impact of this decision on our results. Hereafter, we generally use “terms” to refer to “ N -grams”. For each term, we compute its term and document frequencies, as described below. In our work, we extract term features within daily time intervals, i.e. each day is regarded as a separate time interval. Let D be the set of all news documents and D_t be the set of documents published within time interval t ($D_t \subseteq D$). First, we get $D_t(w)$ as the set of documents containing term w , published within time interval t :

$$D_t(w) = \{d | w \in d, d \in D_t\} \quad (1)$$

The term frequency of w in interval t is computed by:

$$TF_t(w) = \sum_{\forall d \in D_t(w)} f_{w,d} \quad (2)$$

where $f_{w,d}$ denotes the number of occurrences of w in the title and summary of document d . To boost the importance of terms in the title, we suggest to use the weighted version of term frequency in which the frequency of terms in the title is multiplied by a constant factor c , i.e.:

$$WTF_t(w) = \sum_{\forall d \in D_t(w)} (c \times f_{w,d}^{title} + f_{w,d}^{summary}) \quad (3)$$

where $f_{w,d}^{title}$ and $f_{w,d}^{summary}$ are the number of times w appears in the title and summary of d , respectively. Based on our experiments, we use $c = 2$. The document frequency of w within interval t is simply computed by the following relation in which $|D|$ refers to the size of set D :

$$DF_t(w) = |D_t(w)| \quad (4)$$

We also extract a click feature for each term w during interval t . Let C_t be the set of documents clicked during interval t . We compute this set, according to the log file of clicks made on news pages. Each record in this file contains the time of click, the id of news document on which click has been made, the user id, etc. We only consider clicks made by unique real users to get C_t . More precisely, we omit either duplicate clicks by identical users or spam clicks made by automated bots from our log. Next, we obtain $C_t(w)$ as the set of documents containing term w , clicked within interval t :

$$C_t(w) = \{d | w \in d, d \in C_t\} \quad (5)$$

Finally, we identify terms that contain named entities (i.e. real-world objects, such as persons, locations, organizations, etc) using open-source tools like Polyglot.¹ Since famous people and places play an important role in most real-world events, recognizing named entities is essential for an accurate event detection (Sayyadi and Raschid 2013). Our contribution here includes considering title and summary, calculating weighted term frequency and extracting the click frequency feature.

3.2 Burst detection

We identify bursty terms from the extracted terms using Kleinberg’s method (Kleinberg 2003). Kleinberg’s finite machine includes two states for each terms: q_0 and q_1 , corresponding

¹<https://github.com/abosamoor/polyglot>

to low rate and high rates of emission, respectively. This machine models the rate of terms entering the system. For a term w , a transition between two states indicates a significant change in term frequency of w over time intervals. Kleinberg calculated the probability of state transitions using binomial distribution and reported burst intervals using hidden Markov chain. As stated before, we assume a daily basis for time intervals, i.e. each day is considered a new interval. Initially, the machine is in state q_0 and the emission rate is:

$$f_0 = \frac{TF_t(w)}{|D_t|} \quad (6)$$

When the emission rate increases significantly in the next intervals, the machine enters state q_1 with the rate of at least $f_1 = f_0 * s$, where s is a constant scaling factor. If s is large, the probability of entering the burst interval is reduced. In Section 4, we show the impact of configuring s on our results. The machine gets back to state q_0 , if the emission rate drops again. At the end of this step, bursty terms are identified and stored along with their burst intervals.

We now calculate a feature vector for each identified bursty term. These vectors will be used later to give weights to the graph of terms. For a bursty term w in its burst interval b , we compute $TF.IDF$ (Manning et al. 2009), according to the following formula:

$$TF.IDF_b(w) = (1 + \log(WTF_b(w))) \times \log\left(\frac{|D_b|}{DF_b(w)}\right) \quad (7)$$

Combining the relations in (4), (5) and (6), the feature vector of term w in burst interval b is modeled as the following six-tuple:

$$\langle w, b, D_b(w), C_b(w), TF.IDF_b(w), NE(w) \rangle \quad (8)$$

where $NE(w)$ is a binary function that indicates if w contains a named entity or not (it returns 1, if w include a named entity. Otherwise, it returns 0). There may be multiple burst intervals associated with term w . In such cases, we compute one separate feature vector for each burst.

3.3 Graph construction

We construct an undirected graph between terms in which nodes represent bursty terms and edges indicate co-occurrence of terms in news documents published during burst intervals. In other words, there is an edge between any two terms, if and only if, they both have the same burst interval and have co-occur in the same document during that interval. A bursty term may have multiple burst intervals which means there are multiple nodes corresponding to that term in the graph. We weigh nodes based on the value of $TF.IDF$ feature exists in their feature vectors. Then, in order to reduce the size of the graph, we remove trivial nodes whose weight is smaller than a threshold. The value of this threshold is set to the mean weight of nodes in burst intervals.

Finally, we assign a weight to each edge of the graph. Assume w_i and w_j are two terms sharing the same burst interval b . Let e_{ij} denote the edge between w_i and w_j . $DF_b(w_i \cap w_j)$ indicates the document frequency of both terms, i.e. the ratio of documents published during interval b that contain both terms:

$$DF_b(w_i \cap w_j) = \frac{|D_b(w_i) \cap D_b(w_j)|}{|D_b|} \quad (9)$$

Likewise, $CF_b(w_i \cap w_j)$ shows the click frequency of both terms, i.e. the ratio of clicks made during interval b on documents containing both terms:

$$CF_b(w_i \cap w_j) = \frac{\sum_{\forall d \in C_b(w_i) \cap C_b(w_j)} C_b^d}{|C_b|} \quad (10)$$

where C_b^d is the number of clicks on document d during interval b . In addition, we define a boost function η to encourage edges between named entities:

$$\eta(w_i, w_j) = 1 + \frac{NE(w_i) + NE(w_j)}{2} \quad (11)$$

If both w_i and w_j are named entities, η returns 2, and for the case where just one of them is a named entity, it returns 1.5. Putting last three relations together, the weight of edge e_{ij} is obtained as follows:

$$W_{e_{i,j}} = (\alpha * DF_b(w_i \cap w_j) + \beta * CF_b(w_i \cap w_j)) * \eta(w_i, w_j) \quad (12)$$

where α and β are two hyper-parameters used to tune the importance of document and click frequencies ($\alpha + \beta = 1$). We discuss about the setting of α and β in the next section. Note that both frequency features in (12) are normalized to the range [0, 1].

We refer to the above structure as Weighted Bursty KeyGraph (WebKey), since it is a weighted version of KeyGraph that is comprised of just bursty terms, as opposed to the original structure which holds all terms.

Because of using bursty terms and pruning trivial nodes, our WebKey is much smaller than KeyGraph. However, it is still possible that the number of identified bursty terms is very large. In such cases, the WebKey may become too large and complex for processing. Graph sampling is a well-known technique that can help to reduce the size of graph, while still preserving the main properties of base graph. Different methods have been proposed for graph sampling (Leskovec and Faloutsos 2006). In our work, we use the simple RandomNode algorithm (Leskovec and Faloutsos 2006) that randomly selects a node from the base graph and adds it to the sample set. The algorithm repeats this step until the ratio of nodes in the sample set (compared with nodes in the initial graph) reaches a threshold ρ . Finally, it returns a sub-graph of the base graph which is composed of nodes in the sample set and the edges between them.

3.4 Event detection

We use a modified version of edge betweenness centrality on WebKey graph to detect communities. Each community is composed of a set of relevant bursty terms that together represent an event or a topic. First, connected components are found using the Breadth First Search (BFS) algorithm (Cormen et al. 2009a) and then shortest paths between all pairs of nodes are calculated in each component. Unlike previous works, we use the Dijkstra's algorithm (Cormen et al. 2009b) to find shortest paths, because our graph is weighted. Betweenness centrality for an edge is defined as the number of shortest paths passing through that edge (Sayyadi and Raschid 2013). An edge with the highest centrality score is an intercommunity edge over which many paths go. Hence, it can be removed from the graph to distinguish communities. The algorithm keeps removing intercommunity edges until there is no edge with a centrality score higher than threshold θ . This

threshold is calculated as a logarithmic function of the size of WebKey graph G (Sayyadi and Raschid 2013):

$$\theta = m \times \log \left(\frac{|V|(|V| - 1)}{2} \right) + 1 \quad (13)$$

where $|V|$ is the set of nodes in G and m is a constant factor (it is experimentally set to 0.4). At the end of this step, there will be some connected components or communities remaining in the graph. Each community represents an event and the nodes inside that community describe the event. Algorithm 1 presents the pseudo-code of the above algorithm.

Algorithm 1 Event detection algorithm.

```

/* G: WebKey graph (input) */
/* CC: Connected Components */
/* maxEdge: Edge with maximum betweenness centrality */
/* Betweenness[]: Betweenness centrality score for each node */
/* SP: Set of shortest paths */
/* Communities: Detected Events (output) */
1 foreach node  $v$  in  $G$  do
2   |  $Betweenness[v] = 0$ ;
3 end
4  $CC = \text{FindConnectedComponents}(G)$ ; // using Breadth-First Search (BFS)
5 foreach Connected Component  $C$  in  $CC$  do
6   | foreach node  $v$  in  $C$  do
7     | |  $SP = \text{Dijkstra}(C, v)$ ; // returns the set of shortest paths from  $v$  to
8     | | other nodes in  $C$ 
9     | | foreach path  $p$  in  $SP$  do
10    | | | foreach edge  $e$  in  $p$  do
11    | | | |  $Betweenness[e]++$ ;
12    | | | end
13    | | end
14   |  $maxEdge = \text{FindMaxBetweenness}(C)$ ; // find edge with maximum betweenness
15   | score in  $C$ 
16   | if  $Betweenness[maxEdge] < \theta$  then
17   | |  $Communities.add(C)$ ;
18   | |  $G.remove(C)$ ; // remove all nodes and edges in  $C$  from  $G$ 
19   | | if  $|G| == 0$  then
20   | | | return  $Communities$ ;
21   | | end
22   | else
23   | |  $G.remove(maxEdge)$ ; // remove maxEdge from  $G$ 
24   | | goto 1;
25   | end

```

The time complexity of running Dijkstra's algorithm is $O(|E| + |V|\log|V|)$, if Fibonacci heap is used (Cormen et al. 2009b). In the worst case, we have to run this algorithm $|E|$ times for each node. This case happens when we have only one connected component in the WebKey graph and the graph does not partition when we remove the edge with maximum betweenness centrality. Thus, the worst case running time of Algorithm 1 is $O(|E|^2|V| + |E||V|^2\log|V|)$. This worst case is however very unlikely to happen, because: 1) the initial WebKey is usually partitioned, since it includes many irrelevant topics, and 2) due to the notion of betweenness centrality, the graph gets more partitioned after removing edges with maximum centrality score. Therefore, the actual running time is far below the worst case. We experimentally analyze the running time in Section 4.

Figure 2 depicts two sample communities detected by the above algorithm in our dataset in which Persian terms are translated into English. One event is about an airplane crash accident and the other one tells about purchasing airplanes by Iran. Although both communities share a common term (i.e. Bombardier), they are about two separate events.

3.5 Online extension of WebKey

In an online setting where documents are streaming, new nodes and edges are continuously added to the graph, namely the graph is evolving. It is desirable to avoid re-computation of betweenness centrality score for all edges when a new edge is added. There are several researches on online community detection which incrementally compute or approximate the betweenness centrality (Nasre et al. 2014; Kourtellis et al. 2015). Most of them are based on the Brandes algorithm (Brandes 2001) that uses the notion of dependency accumulation along with some auxiliary data structures to reduce running time and memory usage. However, the running time of such algorithms is still prohibitive for online detection of communities in weighted graphs, as the linear BFS algorithm cannot be used any more to compute shortest paths.

Here, we propose an online extension to our method which works based on the simple and fast online community detection algorithm proposed in Zhang et al. (2013). First of all, we use a shorter time interval to extract bursty terms. Based on temporal patterns of our data, we partition each day into two different-sized intervals: cold interval, from 12am till 9am, and hot interval, from 9am till the end of the day. For burst detection, we compute the emission rate of terms in each interval using (6) and compare it with the emission rate in the previous corresponding interval. For example, if the current interval is cold, we compare the emission rate of term w with its rate in the past cold interval. This is because events

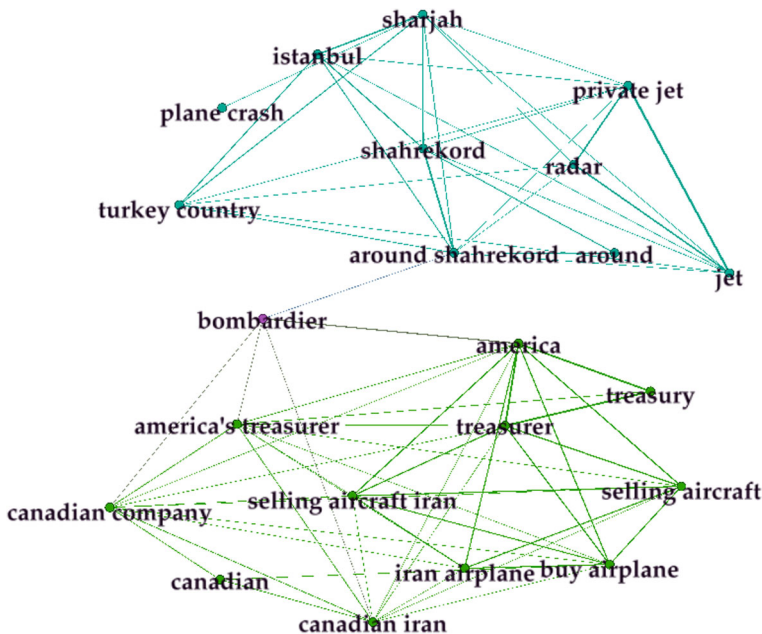


Fig. 2 Two sample communities detected in our dataset

happening in a cold interval usually have a smaller coverage and term frequency than those belonging to a hot interval. It is very likely that a term detected bursty in the current cold interval, is also identified as bursty in the next hot interval. But, we can recognize it much sooner through differentiating cold and hot intervals. To avoid from over-detection of bursty terms, we use a larger scaling factor in cold intervals. In general, we remove non-bursty terms from our graph after one extra interval.

Our algorithm consists of two offline and online steps: 1) finding initial communities using the WebKey method, and 2) incrementally updating the initial communities or adding new communities to them using the online algorithm in Zhang et al. (2013). The first step is done in an offline manner, e.g. with the whole data gathered during the past day. Throughout the next day, we only use the online algorithm to update the initial set of communities. Assume that we have already computed the initial set. When a new document d arrives, we first extract its bursty terms. For each pair of bursty terms in d (like, w_i and w_j), an edge (like, e_{ij}) is considered for addition to the initial graph, if its weight in (12) is greater than a minimum threshold. We have four different cases regarding this edge:

1. both nodes of e_{ij} belong to the same existing community: we do nothing as the new edge just increases the number of intra-community edges.
2. only one node of e_{ij} belongs to the existing graph: assume that w_i is the existing node, while w_j is the new one. We split w_j to a new community with probability inversely proportional to the degree of the existing node w_i in the graph. Otherwise, w_j joins the same community of node w_i .
3. both nodes of e_{ij} belong to the existing graph, but they belong to different communities: we move one node to another community according to the maximum modularity gain principle (Zhang et al. 2013).
4. neither node of e_{ij} belongs to the existing graph: we just assign both to a new community.

The above algorithm has $O(m)$ time complexity and $O(nk)$ memory space complexity where m is the number of edges, n is the number of nodes and k is the number of communities in the graph (Zhang et al. 2013). At the end of each day, we update the initial set of communities by running the WebKey method on the data obtained from that day. The main advantage of the offline step is to merge similar communities and also reduce the number of false positives/negatives via obtaining more precise values for node degree and modularity gain factors which are used in case 2 and 3 of the above online algorithm. We also merge communities that have at least a certain number of inter-community edges between their nodes.

4 Evaluation

In this section, we first discuss the data and also the evaluation model, and then report the results of our evaluations. We also compare our proposed method against famous KeyGraph (Sayyadi and Raschid 2013) and BINets (Ge et al. 2016) methods. Our experiments are conducted on a server with Intel Xeon E5-2699 CPU and 16GB RAM. To speed up the event detection process, we use in-memory Redis² database to store and retrieve nodes and edges in graphs.

²<https://redis.io>

4.1 Experimental dataset

The dataset for this research is collected from the log files of news service in an Iranian local search engine, Parsijoo.³ The majority of users working with this search engine is from the domestic Intranet, sending their queries in Persian language. Parsijoo crawls, parses and stores daily news from more than 100 Persian news websites. We use two types of crawl and click log files in our work. The records in the crawl log file include document id, publishing time, news agency, title, summary, and body of news documents, while the records in the click log file contain click time, document id, user id, and IP address. The first time a user visits the search engine, a unique user ID is assigned to him and then stored in the cookie of his browser. The user ID never expires, unless the user cleans his cookies. There is no record belonging to DDoS attacks in the log, as these attacks are abandoned by the front firewall. Moreover, we remove spam clicks from our log as much as possible (Shakiba et al. 2018).

The crawled dataset contains 132,315 documents from March 1 to 26 (2018) with the total size of 287MB. We split the dataset into two validation and test parts. We use the validation set for tuning our hyper-parameters and constructing the initial WebKey graph and the test set for evaluation and comparison purposes. The click dataset belongs to the same period of time, including 1,689,402 clicks from about 52,100 unique users.

4.2 Evaluation model

We adopt the evaluation model proposed by Petkos et al. (2014). First, we extract events from our dataset. To do so, we investigate daily news manually and if a significant number of documents cover an issue, we consider it an event. To speed up the investigation process, we first index the whole content of news documents in order of their publishing time and then for each document, we find the most similar previously-published documents to discover the coverage of its topic. Due to manual investigation of news documents, this process is a lengthy task and cannot be used as a practical event detection method. Overall, 106 events are extracted as the ground truth. Then, according to the base model (Petkos et al. 2014), three groups of terms are defined for each event in the ground truth:

- *Mandatory terms*: This group includes all terms that are required in a candidate event in order to count it as a correctly detected event.
- *Optional terms*: This group includes a set of terms that are relevant to the event but are not necessary for the correct detection of that event.
- *Forbidden terms*: This group includes a set of terms that should not be included in the candidate event.

For example, for “*Kermanshah earthquake*” event (Kermanshah is a province in Iran), the groups are:

- *Mandatory*: {Kermanshah, earthquake}
- *Optional*: {Richter, magnitude, quake, shake}
- *Forbidden*: {Bushehr, Hormozgan}

There are two other earthquakes around that time in Bushehr and Hormozgan, two other provinces in Iran; therefore, we add these provinces as forbidden terms for this specific event.

³<http://parsijoo.ir>

The events in the ground truth along with their groups of terms are used to evaluate the proposed method and compare it against other methods. We use the following evaluation metrics:

- *Recall*: the percentage of the ground truth events detected by the method, i.e. the portion of ground truth events that all their mandatory terms are returned by the method, while none of their forbidden terms are included in the output.
- *Precision*: the percentage of returned events that are correctly detected by the method (or the portion of returned events that are in the ground truth).
- *F1 measure*: the harmonic mean of precision (P) and recall (R), i.e.:

$$F1 = 2 \times \frac{P \times R}{P + R} \quad (14)$$

In the following two subsections, we concentrate on the results of the original WebKey method. In the last subsection, we discuss the results of the proposed online extension.

4.3 Parameter tuning

As stated before in Section 3.2, there is an adjustable parameter s in Kleinberg's burst detection method. Figure 3 shows the results of three evaluation metrics obtained with different values of s . Moreover, Fig. 4 depicts the running time of our algorithm versus s . As much as we increase s , the burst detection condition becomes tighter and thus, fewer bursty terms and events are identified. As a result, we get a better result for the precision, while the recall worsens. Also, a lower running time is observed for larger values of s , since the size of our WebKey graph is reduced considerably as s grows. For example, the number of bursty terms detected with $s = 3$ is about 40% of the one with $s = 1.5$. Based on these results, we use $s = 2$ for the rest of our experiments, as it yields the highest F1 value. The results in Fig. 3 are achieved when both α and β parameters in formula (12) are set to 0.5. We get nearly the same trend of results for other values of α and β though. To tune α and β hyper-parameters in the weight function of formula (12), we perform an exhaustive search from 0 to 1 with step size of 0.1 for both parameters. Table 1 presents some of the obtained results. The best recall and F1 is achieved at $\alpha = 0.4$ and $\beta = 0.6$. In this case, the recall is about 13% higher than two extreme cases where either α and β is zero. For the rest of

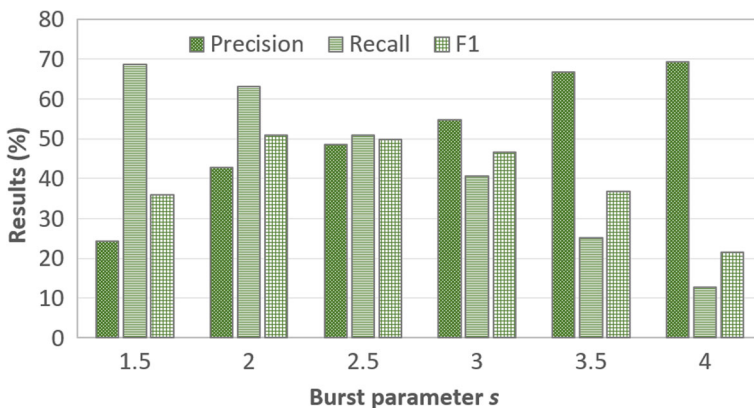


Fig. 3 The evaluation results of our algorithm with different values of burst parameter s

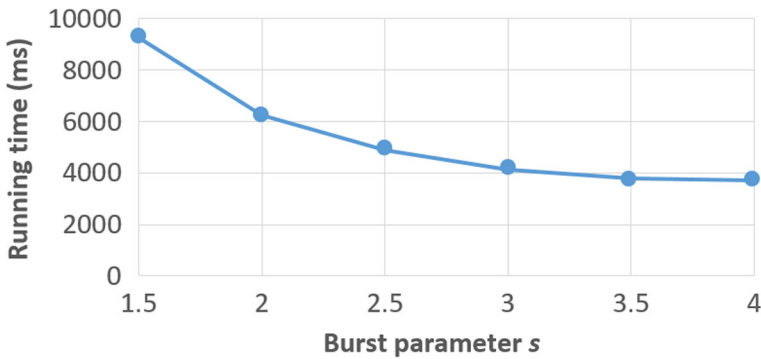


Fig. 4 The running time of our algorithm with different values of burst parameter s

our experiments, we use these two values. From Table 1, it seems that user clicks have a more impact on the quality of event detection, compared with the document frequency factor. However, considering the fact that most of nodes are removed from the WebKey graph based on the minimum $TF.IDF$ threshold, the remaining nodes generally have a high document frequency. As a result of removing trivial nodes, the sensitivity of our algorithm to the setting of α and β is lessened as well. As the third part of formula (12), the function η for boosting named entities can successfully improve the evaluation results by about 5%.

We should assert that the user click factor alone is not a sufficient measure for detecting events. Each day, a lot of hot news are published which can attract the attentions of many users for further clicks, but none of them are events. For example, most of news around celebrities are of this type (e.g. a famous actress attending the concert of a popular singer). On the other hand, there are several news about some important events (e.g. news about an ecological disaster) which absorb few user clicks. In such cases, an event can be correctly detected merely by extracting the co-occurrence graph of bursty terms and integrating the click and document frequency factors together.

We also investigate the impact of sampling threshold ρ on the performance of our algorithm. Figure 5 illustrates the results of three evaluation metrics obtained with different values of ρ . Similarly, Fig. 6 shows the running time and memory usage of our algorithm which we measure during graph construction and event detection steps for the whole dataset. With increasing ρ , the CPU and RAM usages are improved substantially, while the metrics are reduced gradually. For example, by removing 20% of nodes (i.e. $\rho = 80\%$), both running time and memory usage are improved by up to 35%, while the F1 is reduced only

Table 1 The evaluation results of our algorithm with different values of α and β

α	β	Precision	Recall	F1
1	0	0.45	0.6	0.51
0.7	0.3	0.46	0.62	0.52
0.6	0.4	0.46	0.62	0.52
0.5	0.5	0.43	0.63	0.51
0.4	0.6	0.44	0.68	0.53
0.3	0.7	0.43	0.66	0.52
0	1	0.39	0.61	0.48

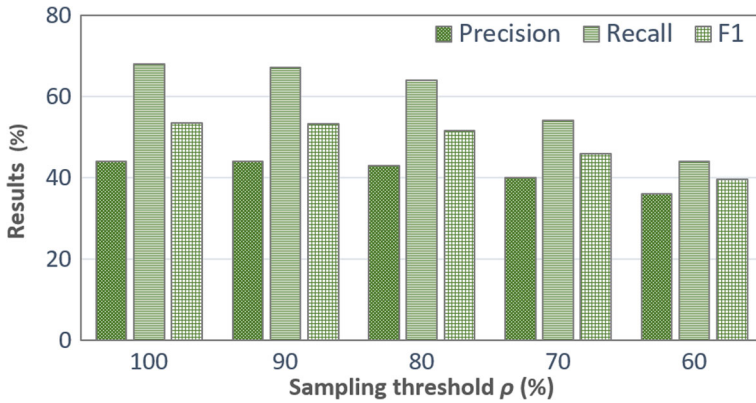


Fig. 5 The evaluation results of our algorithm with different values of sampling threshold ρ

4%. This is due to the fact that there are multiple bursty terms associated with each event and an event can be still detected even if a number of its involved nodes are removed from the graph. However, as a result of eliminating numerous nodes from the graph, some events cannot be distinguished any more, while no new event is detected. Therefore, we observe a rather significant drop in the recall for smaller values of ρ , whereas the precision does not change much. Based on these results, we use $\rho = 80\%$ for our future experiments.

4.4 Comparing algorithms

Finally, we compare the proposed WebKey method with two other famous event detection methods, KeyGraph (Sayyadi and Raschid 2013) and BINets (Ge et al. 2016). The results of this comparison are shown in Fig. 7. WebKey evidently overcomes other two methods, with respect to all three metrics. For example, the precision achieved by our method is 60% higher than that of the closest rival, i.e. BINets. Despite this strong improvement in precision, WebKey also gets a better recall than BINets (overall, F1 is improved by about 40%). In other words, not only more events are detected correctly, but also the number of false events is reduced by our method. We should assert again that KeyGraph does not pay any

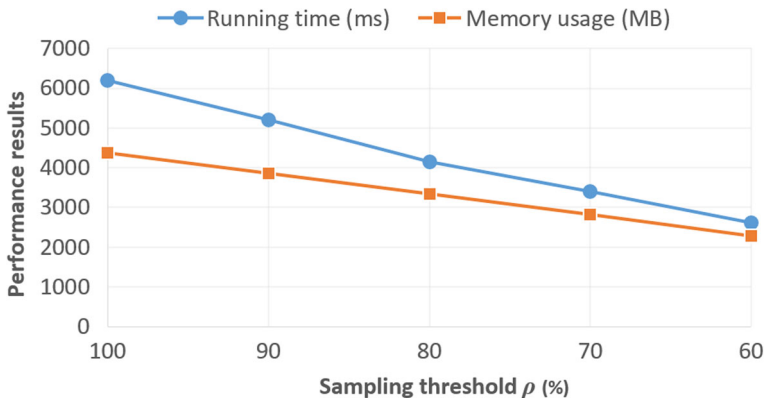


Fig. 6 The performance results of our algorithm with different values of sampling threshold ρ

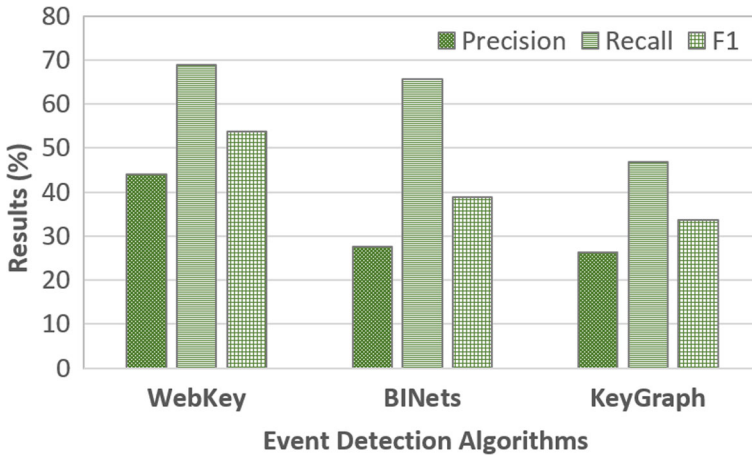


Fig. 7 The evaluation results of different event detection algorithms

attention to bursty terms and moreover, BINets does not differentiate between the importance of nodes and edges in the graph. Hence, they cannot compete with our WebKey which weighs co-occurrence graph of bursty terms using an efficient combination of click and document frequencies. From the viewpoint of running time, KeyGraph is the worst algorithm. Since this algorithm does not identify bursty terms, its graph becomes very large and consequently, the graph construction and event detection steps take a lot of time and space. On the other hand, BINets uses the PageRank algorithm to compute score of nodes, which is time consuming for large graphs. As discussed before, our algorithm employs a number of graph reduction techniques, such as graph sampling, to make a good balance between precision and running time. When we consider only the title and summary of news (instead of the whole news body), the size of WebKey graph is reduced by a factor of 10, whereas the F1 worsens at most 1%. Put another way, the terms in title and summary are sufficient to recognize and express news events. Moreover, the running time is improved about 10% as we remove trivial nodes from the graph, without negatively influencing precision and recall parameters. Overall, our algorithm is on average twice and ten times faster than BINets and KeyGraph, respectively.

4.5 Online extension

For evaluating the performance of our online extension, we simulate news streams through injecting all documents into the system one-by-one in order of their publish time. Based on the result of our experiments, we set the scaling factor s as 4 and 2 for cold and hot intervals, respectively. The response time of our online method for processing a new document is on average below 0.1 seconds which is much smaller than the running time of the WebKey method which lasts more than 6 seconds on 1-day data. More interestingly, the detected communities and their corresponding events are identified in less than 4 minutes from the arrival of the first document describing that event. As a result of using cold and hot intervals, we could identify the events happened during cold intervals about 2 hours sooner, compared with the case where we use the same 1-day intervals.

Table 2 shows the results of the online extension, in comparison with the results obtained from the WebKey method. Note that it is unfair to compare these two algorithms, as they receive different volumes of input data to work. We here just aim to analyze the impact

Table 2 The evaluation results of the online extension, compared with the WebKey method

Algorithm	Precision	Recall	F1
WebKey	0.44	0.69	0.54
Online extension	0.38	0.67	0.48

of using the online extension on the proposed algorithm. Although the precision worsens by about 14%, but we observe only 3% decrease in recall. Precisely speaking, the online extension failed to detect 2 events which had been correctly identified by the WebKey (out of total 106 events). Most of false positives are due to the wrong detection of events in the cold intervals, as we use a looser burst detection during such intervals. It is clear that the WebKey method can better identify events, as it considers the whole graph of bursty terms and their weights and also the betweenness centrality score to detect communities. However, the result of online algorithm is still promising, in a sense that it yields a comparable recall with a much lower running time and memory space which is vital for a practical event detection method. We can also improve the performance of the online method by running the offline WebKey in a shorter time interval to construct the initial graph, since it has a small running time. For instance, the precision increases by about 3% and 8% when we use 12-hour and 1-hour intervals (instead of 1-day interval) to run WebKey, respectively. However, the recall does not change much in either case.

5 Conclusion

In this paper, we propose a fast and efficient method, called WebKey, to detect events in news websites. Our method is based on the detection of bursty terms from news documents and then, constructing a new co-occurrence graph between them. As opposed to previous works, we weigh our graph using important features, such as click and document frequency of co-occurred terms. We also suggest various techniques to improve both accuracy and running time, including: using the title and summary of documents to detect bursty terms, boosting terms in the title, removing trivial nodes, and applying graph sampling to reduce the size of graph. The results of our experiments on a real dataset collected from more than 100 online news websites indicate that our method outperforms a couple of well-known methods by at least 40% in terms of F1 measure. Moreover, it reduces the running time by a factor of at least 2.

References

- Aggarwal, C.C., & Subbian, K. (2012). Event detection in social streams. In *Proceedings of the 2012 SIAM international conference on data mining* (pp. 624–635).
- Allan, J. (2002). Topic detection and tracking: event-based information organization. In *Topic detection and tracking: event-based information organization* (pp. 1–16). Springer Science and Business Media.
- Allan, J., Carbonell, J., Doddington, G., Yamron, J., Yang, Y., et al. (1998). Topic detection and tracking pilot study: final report. In *Proceedings of the DARPA broadcast news transcription and understanding workshop* (pp. 194–218).
- Atefeh, F., & Khreich, W. (2015). A survey of techniques for event detection in Twitter. *Computational Intelligence*, 31(1), 132–164.
- Becker, H., Naaman, M., Gravano, L. (2010). Learning similarity metrics for event identification in social media. In *Proceedings of the 3rd ACM international conference on web search and data mining* (pp. 291–300).

- Borsje, J., Hogenboom, F., Frasinca, F. (2010). Semi-automatic financial events discovery based on lexico-semantic patterns. *International Journal of Web Engineering and Technology*, 6(2), 115–140.
- Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2), 163–177.
- Cataldi, M., DiCaro, L., Schifanella, C. (2010). Emerging topic detection on twitter based on temporal and social terms evaluation. In *Proceedings of the 10th international workshop on multimedia data mining* (Article No. 4).
- Chen, Q., Guo, X., Bai, H. (2017). Semantic-based topic detection using Markov decision processes. *Elsevier Neurocomputing*, 242, 40–50.
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2009a). Breadth-first search. In *Introduction to algorithms*. 3rd edn. (pp. 594–602): The MIT Press. Chapter 22.
- Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. (2009b). Dijkstra's algorithm. In *Introduction to algorithms*. 3rd edn. (pp. 658–662): The MIT Press. Chapter 24.
- Dai, X., & Sun, Y. (2010). Event identification within news topics. In *Proceedings of IEEE international conference on intelligent computing and integrated systems (ICISS)* (pp. 498–502).
- Fung, G.P.C., Yu, J.X., Yu, P.S., Lu, H. (2005). Parameter free bursty events detection in text streams. In *Proceedings of the 31st international conference on very large data bases (VLDB)* (pp. 181–192).
- Garg, M., & Kumar, M. (2018). TWCM: Twitter word co-occurrence model for event detection. *Elsevier Procedia Computer Science*, 143, 434–441.
- Ge, T., Cui, L., Chang, B., Sui, Z., Zhou, M. (2016). Event detection with burst information networks. In *Proceedings of 26th international conference on computational linguistics: technical papers* (pp. 3276–3286).
- Hu, L., Zhang, B., Hou, L., Li, J. (2017). Adaptive online event detection in news streams. *Elsevier Knowledge-Based Systems*, 138, 105–112.
- Kleinberg, J. (2003). Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4), 373–397.
- Kourtellis, N., Morales, G.D.F., Bonchi, F. (2015). Scalable online betweenness centrality in evolving graphs. *IEEE Transactions on Knowledge and Data Engineering*, 27(9), 2494–2506.
- Leskovec, J., & Faloutsos, C. (2006). Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 631–636).
- Li, H., & Yamanishi, K. (2000). Topic analysis using a finite mixture model. In *Proceedings of the joint SIGDAT conference on empirical methods in natural language processing (EMNLP) and very large corpora* (pp. 35–44).
- Li, Z., Wang, B., Li, M., Ma, W.Y. (2005). A probabilistic model for retrospective news event detection. In *Proceedings of the 28th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 106–113).
- Manning, C.D., Raghavan, P., Schütze, H. (2009). *Introduction to information retrieval*. Cambridge University Press.
- Miller, G.A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11), 39–41.
- Mori, M., Miura, T., Shioya, I. (2006). Topic detection and tracking for news web pages. In *Proceedings of the 2006 IEEE/WIC/ACM international conference on web intelligence* (pp. 338–342).
- Nasre, M., Pontecorvi, M., Ramachandran, V. (2014). Betweenness centrality, incremental and faster. In *Springer international symposium on mathematical foundations of computer science* (pp. 577–588).
- Petkos, G., Papadopoulos, S., Aiello, L., Skraba, R., Kompatsiaris, Y. (2014). A soft frequent pattern mining approach for textual topic detection. In *Proceedings of the 4th international conference on web intelligence, mining and semantics (WIMS)* (Article No. 25).
- Phuvipadawat, S., & Murata, T. (2010). Breaking news detection and tracking in Twitter. In *Proceedings of the IEEE international conference on web intelligence and intelligent agent technology (WI-IAT)* (pp. 120–123).
- Sakaki, T., Okazaki, M., Matsuo, Y. (2010). Earthquake shakes Twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World Wide Web (WWW)* (pp. 851–860).
- Sakaki, T., Okazaki, M., Matsuo, Y. (2013). Tweet analysis for real-time event detection and earthquake reporting system development. *IEEE Transactions on Knowledge and Data Engineering*, 25(4), 919–931.
- Sankaranarayanan, J., Samet, H., Teitler, B.E., Lieberman, M.D., Sperling, J. (2009). Twitterstand: news in tweets. In *Proceedings of the 17th ACM SigSpatial international conference on advances in geographic information systems* (pp. 42–51).
- Sayyadi, H., & Raschid, L. (2013). A graph analytical approach for topic detection. *ACM Transactions on Internet Technology*, 13(2), Article No. 4.

- Sayyadi, H., Hurst, M., Maykov, A. (2009). Event detection and tracking in social streams. In *Proceedings of international AAAI conference on web and social media*.
- Shakiba, T., Zarifzadeh, S., Derhami, V. (2018). Spam query detection using stream clustering. *Springer World Wide Web*, 21(2), 557–572.
- Taghi-Zadeh, H., Sadreddini, M.H., Diyanati, M.H., Rasekh, A.H. (2017). A new hybrid stemming method for persian language. *Digital Scholarship in the Humanities*, 32(1), 209–221.
- Wartena, C., & Brussee, R. (2008). Topic detection by clustering keywords. In *Proceedings of the IEEE computer society DEXA workshops* (pp. 54–58).
- Wei, Y., Singh, L., Buttler, D., Gallagher, B. (2018). Using semantic graphs to detect overlapping target events and story lines from newspaper articles. *International Journal of Data Science and Analytics*, 5(1), 41–60.
- Weng, J., & Lee, B.S. (2011). Event detection in Twitter. In *Proceedings of the international AAAI conference on web and social media (ICWSM)* (pp. 401–422).
- Xiaomei, Z., Jing, Y., Jianpei, Z. (2018). Sentiment-based and hashtag-based Chinese online bursty event detection. *Springer Multimedia Tools and Applications*, 77(16), 725–750.
- Yang, Y., Pierce, T., Carbonell, J. (1998). A study of retrospective and on-line event detection. In *Proceedings of the 21st annual international ACM SIGIR conference on research and development in information retrieval* (pp. 28–36).
- Yang, Y., Carbonell, J.G., Brown, R.D., Pierce, T., Archibald, B.T., Liu, X. (1999). Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems and Their Applications*, 14(4), 32–43.
- Yang, C.C., Shi, X., Wei, C.P. (2009). Discovering event evolution graphs from news corpora. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 39(4), 850–863.
- Zhang, W., Pan, G., Wu, Z., Li, S. (2013). Online community detection for large complex networks. In *Proceedings of the 23th international joint conference on artificial intelligence (IJCAI)* (pp. 1903–1909).
- Zhao, W.X., Chen, R., Fan, K., Yan, H., Li, X. (2012). A novel burst-based text representation model for scalable event detection. In *Proceedings of the 50th annual meeting of the association for computational linguistics: short papers* (pp. 43–47).

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.