Finding Story Chains in Newswire Articles

Xianshu Zhu
CSEE Department
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore, MD USA
Email: xianshu1@umbc.edu

Tim Oates
CSEE Department
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore, MD USA
Email: oates@cs.umbc.edu

***Abstract*—Massive amounts of information about news events are published on the Internet every day in online newspapers, blogs, and social network messages. While search engines like Google help retrieve information using keywords, the large volumes of unstructured search results returned by search engines make it hard to track the evolution of an event. A story chain is composed of a set of news articles that reveal hidden relationships among different events. Traditional keyword-based search engines provide limited support for finding story chains. In this paper, we propose a random walk based algorithm to find story chains. When breaking news happens, many media outlets report the same event. We have two pruning mechanisms in the algorithm to automatically exclude redundant articles from the story chain and to ensure efficiency of the algorithm. Experimental results show that our proposed algorithm can generate coherent story chains without redundancy.**

I. Introduction

Nowadays, the flood of information on the Internet can easily swamp people, which seems to produce more pain than gain. While there are some excellent search engines, such as Google, Yahoo and Bing, to help us retrieve information by simply providing keywords, the problem of information overload makes it hard to understand the evolution of an event. Conventional search engines display unstructured search results. The search results are ranked by relevance, including keyword-based methods of ranking search results, PageRank [1], and other more complicated ranking algorithms. However, when it comes to searching for a story (a sequence of events), none of the ranking algorithms above can help to organize the search results by evolution of the story.

Limitations of unstructured search results include: (1) **Missing the big picture on complex stories**: For complex news stories, users can spend significant time looking through unstructured search results without being able to see the big picture of the story. For instance, Hurricane Katrina struck New Orleans on August 23, 2005 and many news articles related to this hurricane were published from every major media outlet throughout the world every day. By typing ”Hurricane Katrina” in Google, people can get much information about the event and its impact on the economy, education, health, and government policies. However, people may feel desperate to sort the information to form a story chain that tells how, for example, Hurricane Katrina has impacted government policies. We seek to extend the capability of existing search engines to output coherent story chains, rather than loosely connected pieces of information. Previous research in event threading and tracking [2], [3] has largely focused on organizing news articles into hierarchies or graphs, but little effort has been made on presenting search results in a meaningful and coherent manner. Shahaf et al.[4] were the first to address the output coherence problem. However, the algorithm discussed in [4] can be further improved in many ways. Another limitation of unstructured search results is: (2) **Hard to find hidden relationships between two events**: The connection between news events are sometimes extremely complicated and implicit. It is hard for users to discover the connections without thorough investigation of the search results. Such hidden relationships between news events can be very useful for users to obtain a deep understanding of the event, because every news event happens for some reason, and many have impact on various aspects of our lives. The information that the user gets from search engines would be more informative if we could uncover the hidden relationships between news events.

In this paper, we propose a random walk based algorithm that can automatically find story chains. More specifically, our algorithm can find out how two events are correlated by finding a chain of events that coherently connect them together. A good story chain needs to have the following characteristics:

1) Relevance: The articles on the chain should be relevant to the events connecting the two articles.
2) Coherence: The chain should be coherent. The transition between nodes on the chain should be smooth, with no concept jumping or jittering. This allows users to have a better understanding of the progression of the story after reading the chain.
3) Low Redundancy: When breaking news happens, many media outlets report the same event. Users may prefer to read a story chain that contains only one representative article for every event.
4) Coverage: Good story chains should cover every important event of the story.

Moreover, efficiency is an important factor in developing the system of discovering story chains, since no user wants a system that takes a long time to compute a story chain.

Based on the discussion above, our goal is to develop an algorithm that can efficiently find hidden relationships between news events and output a story chain that is coherent and relevant, with high coverage and low redundancy. Shahaf et al. [4] already addressed the coherence and relevance problems.

Our work mainly focuses on the latter two problems (low redundancy and efficiency), while still maintaining a highly coherent and relevant story chain.

The rest of this paper is organized as follows. Section II describes related work. Section III describes the story chain problem. Section IV contains a detailed description of our algorithm for finding story chains. Section V provides experimental results, and Section VI concludes the paper.

II. Related Work

Previous research in event detection has largely focused on grouping retrieved documents into events according to the similarity of their contents and time stamps. They focused on organizing news articles into hierarchies, but little effort was made on presenting data in a meaningful and coherent manner. Shahaf et al. [4] were the first to address this problem, and our work is highly motivated by them. In their work, a method was proposed for automatically finding story chains. They define the notion of chain coherence to be as strong as its weakest link. Then, they formalize the story chain finding problem into a linear programming (LP) problem, with the objective function of maximizing the influence of the weakest link. A set of random walks are simulated on a word-document bipartite graph to calculate $Influence(d_i, d_{i+1}|w)$, which is the influence of document d_i on d_{i+1} through word w. Let $|D|$ be the number of documents, and $|W|$ be the number of words. They need $O(|D|)$ random walks and the LP has $O(|D|^2 \cdot |W|)$ variables. Results show that their method can find coherent chains. The drawbacks of that work are: (1) Efficiency: The time complexity of the algorithm is high. Even though they used some speed-up methods to scale the algorithm, it still takes ten minutes for the creation of a chain of length 6 or 7. (2) Redundancy: The redundancy problem is caused by including multiple articles for a single event in the story chain. They did not address the redundancy problem. In our work, we propose a random walk based method that can handle redundancy and is more computationally efficient.

Fung et al. [5] studied the problem of identifying related keywords given the limited query keywords provided by users. They proposed a Time Driven Documents-partition (TDD) method to construct an event hierarchy in a text corpus based on a given query. However, we enrich the query related features by asking users to provide an article as input. Nallapati et al. [2] investigate methods for event threading. They developed a time-decay based clustering approach for both clustering stories into events and constructing dependencies among them. Mei et al. [3] study the problem of detecting topics from temporal text streams. A topic evolution graph is built and used to trace topic transitions. Chen et al. [6] developed a user interface that organizes Web search results into hierarchical categories. However, none of the methods above tried to categorize documents based on causes and effects.

Timeline generation [7], [8], [9] generates news story evolution tracjectories along a timeline given query related news collections. Yan et al. [9] proposed a framework for Evolutionary Timeline Summarization(ETS). They also considered coherence as a key requirement for timelines. However, our task is different that we do not try to generate summaries. Instead we are trying to re-organize news articles in a more meaningful and coherent manner based on a user's query.

Significant work has been done on random walks on graphs to find associations between two objects. Sun et al. [10] propose a model based on random walks on bipartite graphs to detect neighborhoods and anomalies. Xiang et al. [11] propose a temporal personalized random walk method to capture user's temporal preference. Angelova et al. [12] studied the propagation of labels in web graphs. In this paper, we find the relevance score between two documents based on random walks on a bipartite graph. We further introduce time nodes into the graph to capture the temporal attribute of the documents.

III. Problem definition

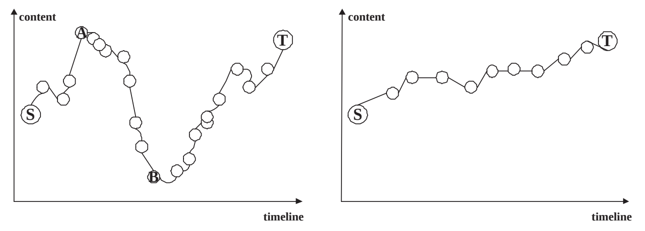
Suppose we have a set of chronologically-ordered articles $d_1, d_2, \cdots, d_n$ obtained by key word search. Users may want to refine the search results by looking at the relationship between two news events. They are allowed to choose two news articles from the search results to be start and end nodes, respectively. The goal is to find a chain of articles that can form a coherent story connecting both the start and end nodes. In figure 1, we plot two sample story chains in a coordinate plane in which the x-axis is time and the y-axis is the content of the article. The story chain shown in figure 1(a) has a big jump between article A and B, which makes the story chain incoherent. Moreover, the chain contains redundant articles, such as those around A in the graph. In contrast, the story chain shown in figure 1(b) has high coherence and relevance and low redundancy. A common and easy way to solve the redundancy problem is to divide the time line into K equally-spaced time bins and choose one article in each of the time bins. However, the development of a story is not usually linear in time. There may be frequent updates about a story in a certain time range, or the pace of story evolution may slow down. Therefore, this method may prune some articles that are essential to the evolution of the story. Moreover, the performance of this method is highly related to the bin size parameter and highly dependent on the nature of the story. It cannot reduce redundancy if the bin size is too small.

Moreover, efficiency is an important factor that we need to consider, since no user wants to use such a system that takes a long time to compute a story chain.

A potential major challenge in this task is branching, namely, the different paths a story can evolve which gives users the options to choose which direction they want to go. However, in this paper we assume users already select a branch (end node) that a story chain should follow. The branching problem will be explored in the future.

IV. Story chain finding algorithm

Our story chain algorithm, as described in Algorithm 1, contains two iterative stages: (1) search for articles that can be added to the chain; (2) prune articles, which includes (a) pruning the least relevant articles and (b) pruning redundant articles. The two stages work iteratively until no more articles



(a) Bad story chain (b) Good story chain

Fig. 1. Story Chain

can be added to the chain. We formalize the story chain finding problem as a divide and conquer bisecting search problem. The initial story chain contains only one link $s-t$, where s is start article and t is end article. Each time we insert a node on a link, the link will be divided into two sub-links. The bisecting search adds a node on each sub-link recursively. The final story chain will be composed of multiple links. We simulate random walks on a bipartite graph to calculate article relevance scores, which are used to select the best nodes to add to the link.

Algorithm 1 Story chain finding algorithm

```
Input: chronologically-ordered articles $d_1, d_2, \cdots, d_n$, Start node $s$, End node $t$
Initialize story chain $C = s$ - $t$, input link $l = \{s$ - $t\}$
repeat
    1. Pruning process (a): Prune least relevant articles
    2. Select a best article $a_i$, that can be added to the link. Story chain becomes $C = s$ - $a_i$ - $t$.
    3. Pruning process (b): Prune redundant articles
    4. Update input link as $l = \{s$ - $a_i, a_i$ - $t\}$. Repeat step 1, 2 and 3 for each of the input link in $l$
until There are no articles left in the set. (Articles are either been added to the chain or have been pruned.)
Output: story chain $C = s$ - $a_1$ - $a_2$ - $\cdots$ - $a_i$ - $t$
```

The advantages of this algorithm are:

- Coherence and relevance: This method can create high quality chains in terms of coherence and relevance. See section IV-B for more details.
- Efficiency: We just need $O(logk)$ random walks to find a chain of length k. This saves computation compared to the method in [4]. Moreover, we can save even more computation by using the two pruning methods, because random walks will be simulated on a much smaller graph.
- Redundancy: Redundant documents are removed while pruning.
- Two pruning methods, which will be discussed in more detail below.

In the following subsections, we describe the two stages as well as how we compute link strength.

A. Compute link strength

Our goal is to find the best articles to be added to a link so as to improve the strength of that link. In this way, we can find a story chain with every link on the chain of high strength. The strength of a link is determined by the correlation of two articles that connect to each other. Intuitively, two articles are correlated if they share many common words. The more words they have in common, the more strongly they are correlated. However, we can not conclude that two articles are not relevant when they don't share any common words. For example, an article which contains words 'storm' and 'tornado' does not share any common words with article which contains word 'hurricane'. However these two articles might be related.

We construct a bipartite graph, as shown in figure 2. Bipartite graph $G = \langle V_D \cup V_W, E \rangle$, where $V_D = \{d_i | 1 \leq i \leq m\}$ and $V_W = \{w_i | 1 \leq i \leq n\}$, $E \subset V_1 \times V_2$. The vertices correspond to documents and words. The edge weights represent the strength of the association between a document and a word. We use TF-IDF weights for document-to-word edges. A short random walk starting from d_i should reach d_j frequently if these two articles are highly relevant to each other. d_a usually has a high relevance score to d_b if (1) direct relevance: they have many shared words. For example, d_1 has many shared words with d_2, see figure 2; or (2) indirect relevance: they both are relevant to a common article. For example, even though d_1 has no shared words with d_3, random walks starting from d_1 can frequently reach d_3, because they both are closely related to d_2, as is shown in figure 2.

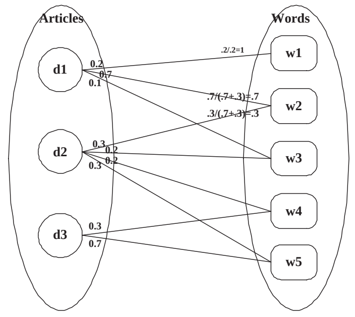


Fig. 2. Method to solve redundancy problem

B. Stage 1: Search

Given start node s and end node t, we want to find a coherent chain linking s and t. Forming a coherent chain is easy when s and t are close. However, the problem becomes very difficult when s and t are far away. Motivated by the idea of divide and conquer, we formalize the story chain finding problem as a bisecting search problem. We divide the chain into two sub-chains by selecting the middle node of the chain and solving the sub-problem recursively. We fist compute $\underset{d_i}{\operatorname{argmax}}\, r_s(d_i) * r_t(d_i)$, where $r_s(d_i)$ is the probability that random walks reach d_i from s. The formula means document d_i, which can be reached most frequently from both nodes, will be put on the story chain. Figure 3 is an illustration of the search algorithm. Node A is selected because it can be frequently reached from both s and t. After step 1, the problem of finding a chain linking s and t is reduced to two sub-problems, which are (1) finding a chain between s and A; (2) finding a chain between A and t. The chain search process is very directional under this search method. To be more specific, it avoids s wandering around lots of irrelevant nodes before going back to t. In other words, this method allows to generate a coherent and relevant story chain.

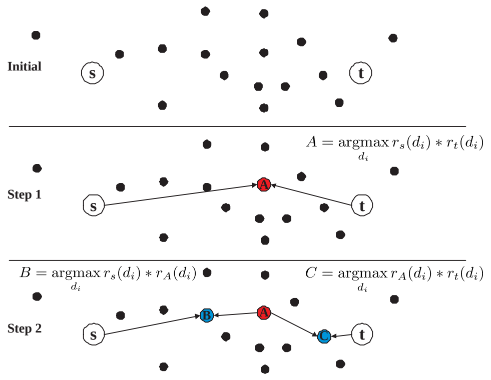

Initial
s
t
Step 1
$A = \underset{d_i}{\operatorname{argmax}}\, r_s(d_i) * r_t(d_i)$
A
Step 2
$B = \underset{d_i}{\operatorname{argmax}}\, r_s(d_i) * r_A(d_i)$
$C = \underset{d_i}{\operatorname{argmax}}\, r_A(d_i) * r_t(d_i)$
B
C

Fig. 3. Illustration of search algorithm

C. Stage 2: Pruning

Simulating random walks on a graph containing thousands of article-nodes and word-nodes is time consuming, even though our binary story chain search algorithm only needs a small set of random walks. We want to further improve the efficiency of the algorithm by pruning unnecessary articles in each recursion. We do two types of pruning before going directly to the next level of search recursion: (1) Prune least relevant articles; (2) Prune redundant articles

1) Prune least relevant articles: A story chain can be viewed as an association rule, where each link in the chain is a subset of the rule. Similarly, the strength of any sub-link in the chain must be larger than or equal to the coherence score of the chain. In other words, the coherence score of the chain can be defined as the weakest link of the chain, which is $Coherence(d_1, d_2, \cdots, d_n) = \min_{i=1\cdots n-1} linkStrength(d_i, d_{i+1})$. We can prune the least relevant articles based on this idea.

In figure 4, we simulate a random walk starting from s and compute the relevance score between s and all the other articles in the graph. Obviously, s is more relevant to t compared to D. If we add D into the chain, the coherence score of the chain will decrease, which is $Coherence(s, D, t) = linkStrength(sD)$. Thus, D should not be included in the chain, even though D is close to t. We can prune all articles d_i such that $r_s(d_i) < r_s(t)$ or $r_t(d_i) < r_t(s)$, where $r_s(d_i)$ and $r_t(d_i)$ are the probabilities that a random walk reaches node d_i from node s and t, respectively. Articles outside the solid arc will be pruned. However, the random walk, which is used to calculate relevance score between articles, is based on word similarity. There might be a gap between content similarity and semantic similarity. Therefore, we modify the threshold to be $r_s(d_i) < r_s(t) - \alpha$ or $r_t(d_i) < r_t(s) - \alpha$ to avoid over pruning. Articles outside the dotted arc will be pruned. α is a parameter which controls how aggressive the pruning should be. In implementation, we simplify this by pruning only a portion of the nodes that are least relevant. For example, we will prune the top 80% of the nodes d_i that satisfy $r_t(d_i) < r_t(s)$ or $r_s(d_i) < r_s(t)$.

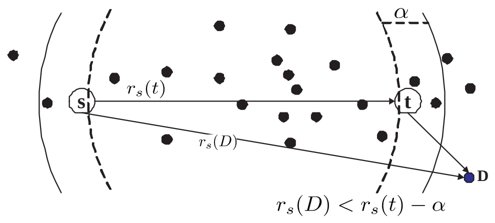


Fig. 4. Prune least relevant articles (r_s are probability values. The larger the r_s value, the closer the relationship between two nodes will be.)

2) Prune redundant articles: When the binary search picks one article, we want to further prune articles similar to that article both in content and time. Since there could be various news outlets reporting the same event, we just want to select a representative article. Moreover, there can be similar events that happen at different times, so we don't want to remove similar articles with different time stamps. This is the intuition of introducing time nodes into the random walk graph.

We add time nodes into the previous bipartite graph. Thus, the graph becomes a tripartite graph with three different kinds of nodes: document nodes, word nodes, and time nodes, as is shown in figure 5. We split the whole time period into multiple equal time bins. The duration of each bin is of p days. The document-to-time weight is always 1, because each document can be associated with only one time stamp. The time-to-document weight is normalized over the number of documents connecting to the time node. For example, there are two articles connecting to time node $t1$. Thus, the time-to-document weight for $t1$ is $1/2 = 0.5$.

After adding time nodes to the graph, the outlink weights for document nodes do not sum up to 1. Without changing the original weight of the graph, we use a hierarchical method to construct the transition matrix for the tripartite graph. Level 1: The transitions starting from document nodes will go to time nodes with probability α and go to word node with probability $1-\alpha$. Level 2: follow the transition probability on the original graph. For example, the transition probability from $d1$ to $w1$ is 0.2, after $d1$ is selected to go to a word node. We can alter the value of α to adjust how influential the time nodes should be.

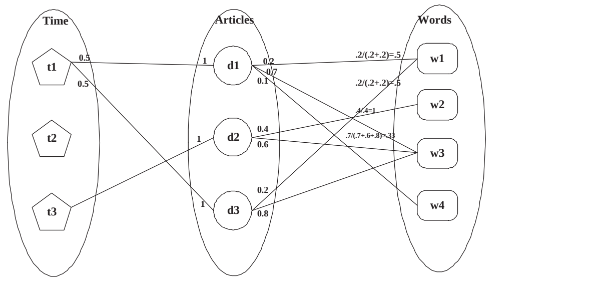


Fig. 5. Method to solve redundancy problem

The new random walk starts from document nodes on the tripartite graph and will be more likely to reach articles that are in the same bin and close in content. One drawback of this graph is it puts more weight on the articles that are in the same time bin and ignores the difference between bins that are close and bins that are far away. We extend the typical random walk algorithm to support reflecting effects of time decay. We assume the random walk starts from d_a. The extended random walk formula is:

$$\vec{r} = (1-\beta)M\vec{r} + \beta\vec{N} \tag{1}$$

where $\vec{N}$ is a vector indicating which nodes the random walk will jump to after a restart.

$$\vec{N}(d) = \begin{cases} \lambda, & d = d_a \\ (1-\lambda)wdecay_i, & d = t_i \\ 0, & otherwise \end{cases} \tag{2}$$

The vector $\vec{N}$ should bias both document nodes and time nodes, which means that the random walk is more likely to jump to its own document node and corresponding time nodes, or time nodes that are close in time after a restart. We choose an exponential decay function $f(n) = exp(-\gamma n)$ for $\gamma > 0$. Let the corresponding time bin for d_a be t_a. k is the duration time of each time bin. The weight $wdecay_i$ that is assigned to each time bin t_i is: $exp(-\gamma|t_a - t_i|)$. Then, the weights $wdecay_i$ are normalized to sum up to 1.

V. Experiments and Evaluation

In the experiments, we aim to answer this fundamental question: can our story chain finding algorithm produce good story chains efficiently? In addition, we also show how different pruning methods affect story chain construction and the efficiency of the algorithm. The quality of story chains is evaluated using four criteria: relevance, coherence, coverage and redundancy. As there are no standard datasets suitable for our task, we constructed our own data sets on real news articles which will be discussed in section V-A in detail.

- **Connecting-Dots[4]**: We cannot compare our results with theirs because we don't have access to their code and the paper lacks implementation detail for us to implement the algorithm by our own. More importantly, in this paper we concentrate on developing a more efficient algorithm while maintaining high quality of the story chain, especially producing story chains with low redundancy. In the experiments, we will show the computational complexity in terms of execution time of the algorithm.
- **Event threading (TDT)** [2]: In this work, a time-decay based hierarchical clustering approach was proposed to cluster news articles into multiple clusters and find dependencies among them. Assume that each cluster represents an event. Then a story chain can be constructed by picking representative articles from each cluster in the dependency path. In section V-C3 we compare our algorithm with the TDT algorithm.

It is a very subjective task to evaluate the quality of the story chain. The evaluation was conducted on Amazon's Mechanical Turk (MTurk)[1]. MTurk is a marketplace where requesters can publish human intelligence tasks (HITs) to multiple workers and workers can choose to complete tasks. We publish our story chains evaluation task to MTurk. Workers who choose to do the task will rank the story chains from best (5) to worst (1) by relevance, coherence, coverage and redundancy.

[1]https://www.mturk.com/mturk/welcome

Topics	# Docs	Year	Query Key Words
O.J.Simpson Trial	10475	06/94-08/97	O.J.Simpson
Hurricane Katrina	3834	08/05-06/07	Hurricane Katrina
Japan Earthquake 2011	6607	03/11-11/11	Japan earthquake 2011; Japan nuclear 2011; Japan tsunami 2011; nuclear disaster 2011; Fukushima Daiichi nuclear disaster

TABLE I
Data Set Information

A. Data Set

We constructed data sets for three news topics: the O.J. Simpson Trial, Hurricane Katrina and the earthquake and tsunami in Japan 2011. The O.J. Simpson Trial was a very publicized criminal trial in American history. Many news articles from various news outlets were published about this event. Over 10,000 articles contain the key word ”O.J. Simpson”. Thus, it is hard for web users to keep a clear picture of the story. On the other hand, Hurricane Katrina and the earthquake in Japan 2011 are both complex stories, since they have different impacts on many aspects, such as the economy, education, health, the environment and government.

We use the ”North American News Text[2]” and the ”New York Times Annotated Corpus[3]” data sets from the Linguistic Data Consortium (LDC) to construct the O.J. Simpson Trial and Hurricane Katrina datasets. Articles are selected to construct datasets based on keyword search. The ”North American News Text” data set contains news articles (1994-1996) from multiple sources, including the Los Angeles Times & Washington Post, New York Times News & Syndicate, Reuters News and Wall Street Journal. The ”New York Times Annotated Corpus” data set contains articles from the New York Times between 1987 to 2007. We also crawled news articles from multiple sources (New York Times, USA Today, Washington Post, Reuters and CNN) to construct a data set on the earthquake in Japan 2011. Table I shows detailed statistics of these three datasets.

For each article in the dataset we convert words to lowercase, remove stop words, and do word stemming. We only use the most frequent words in the dataset for graph construction.

B. Exprimental parameters

Our goal is to construct a story chain representing the evolution of the story given the start (d_s) and end (d_t) articles as input. The method should be able to automatically select only one representative article for each event. We evaluate our method by considering five stories in table II. These five stories contain both simple stories (such as C4) and complex stories (such as C1, C2, C3, and C5). The complexity of a story can be defined as how relevant the start and end articles are. For example, story C1 is a complex story since Hurricane

[2]http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC95T21
[3]http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2008T19

Katrina and government policies are not directly connected. In addition, we also consider the same story but different threads. For example, both story chain C2 and C3 are about the Japanese earthquake, and both story chain C4 and C5 are about the O.J. Simpson trial.

C. *Performance Evaluation*

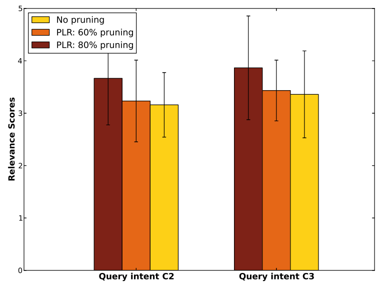
There are two pruning steps in our story chain finding method: (1) Prune least relevant articles (PLR); (2) Prune redundant articles (PR). We show the performance for random walks with PLR only and random walks with PR only, respectively. We also show the performance with both pruning methods (PLR-PR). The performance evaluation results are based on human judgements through MTurk. Story chains are ranked from best (5) to worst (1) by relevance, coherence, coverage and redundancy.

1) Performance of PLR (prune irrelevant articles): In this section, we show the experimental results for PLR. We further change the aggressiveness in PLR to see how it affects the results. Story C2 and C3 both are related to Japan Earthquake 2011. The Japan earthquake dataset spans multiple topics, such as earthquake and tsunami rescue, nuclear crisis, economic loss etc. Thus this data set contains many irrelevant articles as opposed to a specific query. Figure 6(a) shows the relevance score of the story chain increases as we prune more irrelevant articles. The more aggressive the pruning is, the higher the relevance score is. However, in figure 6(b) the coherence score increases as we remove irrelevant articles but the score decreases as the pruning becomes more aggressive. This is because over pruning causes some relevant articles to be removed.

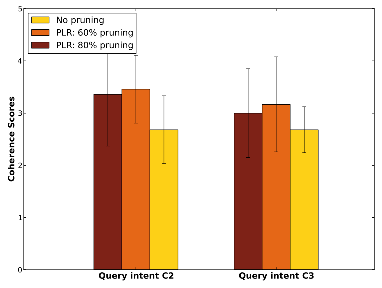
2) Performance of PR (only prune redundant articles): In this section, we show the experimental results for PR. Figure 7 shows that the story chain contains much less redundant articles with PR.

3) Performance of both PLR-PR: Figure 8 compares the performance among three algorithms: Event threading, No-pruning and Both-pruning(PLR-PR). The performance of the event threading method is not as good as the two pruning methods in terms of relevance, coherence, and coverage. On the other hand, the event threading method outperforms the No-pruning methods in terms of redundancy. This is because it groups articles into clusters which to some degree relieves the redundancy problem. However, the size of the clusters is hard to control. Sometimes when the clusters are too big, it will result in low coherence and coverage if we only select one representative article in each cluster. Figure 9 shows the algorithm performance with both pruning methods. We can see that PLR-PR algorithm outperforms random walk with no pruning algorithm on all of four criteria.

4) Computational complexity: The computation overhead of the story chain algorithm includes two parts. The first part is to parse the data set, compute TF-IDF weights and then compute graph edge weight. This is a one time computation. Once the graph is constructed, our algorithm can find different story chains based on different queries (start and end nodes). The second part is the random walk computation and graph weight update. The story chain algorithm can find a story chain in around ten iterations. In each iteration, it needs two random walks from the start and end node respectively. Our algorithm requires only $O(logk)$ random walks to find a story chain of length k, which is much lower compared with the computational overhead of $O(|D|)$ random walks to compute word influence and solve a linear programming problem with $O(|D|^2 \cdot |W|)$ in paper [4], where $|D|$ is total number of articles in the dataset and $k \ll |D|$. Moreover, the graph size is largely reduced with the help of the two pruning method, which further speeds up the random walk computation. Figure 10 shows the number of remaining documents in each iteration. Most of the irrelevant articles are pruned in the first two iterations. It introduces little computational overhead to update graph weights. Since TF-IDF values can be easily updated incrementally, we do not have to recompute the values from



(a) Relevance Score



(b) Coherence Score

Fig. 6. Performance of PLR

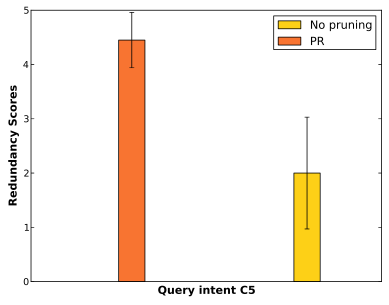
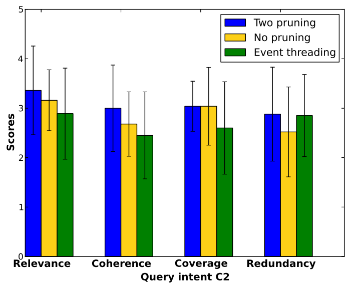


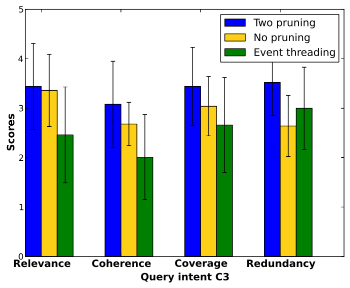
Fig. 7. Performance of PR

Query intent	Start d_s and end d_t articles
C1: How Hurricane Katrina is related to government policies	d_s: Hurricane Katrina hit New Orleans (08/26/05) d_t: Attacking Bush, Clinton Urges Government Overhaul (04/14/07)
C2: How Japan earthquake has impact on nuclear policy in German nuclear company	d_s: Japan super quake, tsunami terrify tremor-prone nation (03/11/11) d_t: E.ON to sue German government over nuclear closure (11/14/11)
C3: How Japan earthquake has impact on competition between Toyota and Volkswagen	d_s: Japan super quake, tsunami terrify tremor-prone nation (03/11/11) d_t: Volkswagen may topple Toyota as world's top automaker (10/24/11)
C4: Story about O.J. Simpson trial	d_s: O.J. Simpson's Ex-Wife Found Dead in Double Homicide (06/13/94) d_t: Simpson jury reaches verdict in six hours (10/02/95)
C5: How O.J.Simpson trial has impact on racial problems	d_s: O.J. Simpson's Ex-Wife Found Dead in Double Homicide (06/13/94) d_t: Race flares anew as polarizing issue in U.S. life (10/19/95)

TABLE II
INITIAL INFORMATION OF THE STORY CHAINS

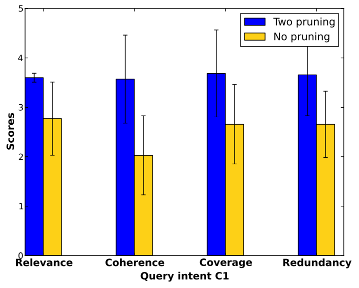


(a) Story chain C2

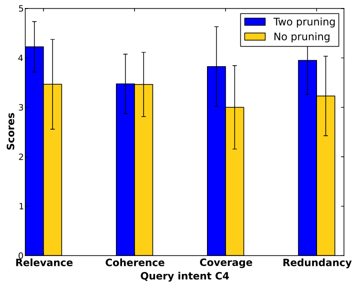


(b) Story chain C3

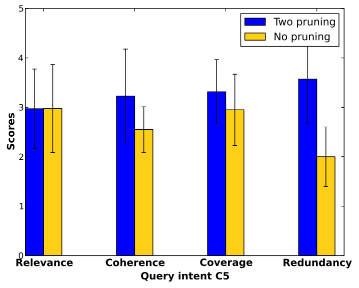
Fig. 8. Performance of PLR-PR, No-pruning and Event threading



(a) Story chain C1



(b) Story chain C4



(c) Story chain C5

Fig. 9. Performance of PLR-PR and No-pruning

the very beginning.

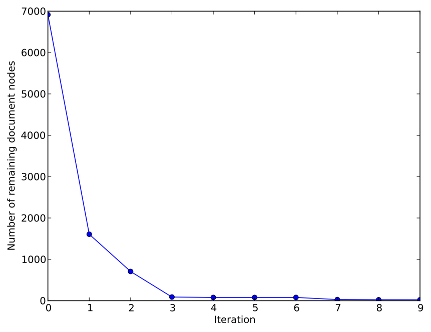


Fig. 10. Number of remaining documents in each iteration

Table III shows the running time of the algorithm. We set the running time for the algorithm without any pruning to be one time unit. The value in the table shows how many time units PLR, PR, and PLR-PR take respectively with respect to no-pruning method. We can see that the running time for the PLR algorithm decreases to only 33% of the time since it removes large numbers of irrelevant articles from the graph which makes random walk computation much smaller. Algorithm PR needs much more running time, because the no-pruning algorithm needs two random walk computation starts from start and end nodes respectively and PR requires one more round of random walk on a tri-partite graph based on the original data set to find redundant articles. The running time for PLR-PR which combined two pruning methods together is on average only 40% of what the no-pruning method requires. The algorithm is running on a laptop with Intel Core i5 2.67GHz and 4G memory. The average running time for the no-pruning algorithm is 5 minutes and the algorithm with

Algorithm	C1	C2	C3	C4	C5
No-pruning	1	1	1	1	1
PLR - 60%	0.51	0.33	0.45	0.11	0.25
PR	1.78	1.79	1.39	1.55	1.77
PLR-PR	0.67	0.47	0.55	0.15	0.38

TABLE III
RUNNING TIME OF THE ALGORITHM

both pruning algorithm reduces the time to only 2 minutes. Since the random walk is basically matrix multiplication, we believe that there are many exsiting methods, such as parallel computing etc, to optimize or speed up matrix multiplication which can further reduce the running time of our algorithm and make it very scalable.

5) Same story, different thread: In this experiment (table IV), we keep the start node the same but change the end node of the story chain. One story ends with the verdict of O.J. Simpson's trial, while the other story ends with discussion of race issues in the U.S. The latter generates a completely different story chain, which focuses on how race issue played an important role in the jury. The result shows that our story chain algorithm is quite adaptive to users' queries. It helps web users to easily understand different aspects of a story.

(a) Story chain on O.J.Simpson trial

(1) O.J. Simpson's Ex-Wife Found Dead in Double Homicide (06/13/94)
(2) Five Days After Ex-Wife's Murder, O.J Simpson Faces Charges (06/17/94)
(3) Simpson Prosecutors Win Right to Begin DNA Tests (07/25/94)
(4) Chronology of events in O.J. Simpson murder case (09/26/94)
(5) Simpson Jurors To Be Sequestered, Starting Wednesday (01/09/95)
(6) Whether police and prosecutors rushed to judge Simpson as a suspect (03/17/95)
(7) Simpson limo driver tells of tall, black figure (03/28/95)
(8) Simpson jury reaches verdict in six hours (10/02/95)

(b) Story chain on how Simpson trial related to race issue

(1) O.J. Simpson's Ex-Wife Found Dead in Double Homicide (06/13/94)
(2) O.J. Simpson Questioned In Ex-wife's Murder (06/14/94)
(3) Simpson's Lawyer Says Police Overlooked, Faked Evidence (01/25/95)
(4) Simpson judge denies trial delay but slams defense (01/30/95)
(5) Tough Math Problem at O.J. Trial (Whether the bloodstain is O.J.'s)(06/22/95)
(6) Lawyers Argue Over Simpson Glove Evidence (08/01/95)
(7) Prosecution Gears Up for Rebuttal in Simpson Case (Mark Fuhrman and the taped interviews) (09/10/95)
(8) Defense makes final bid for Simpson's acquittal (mention Fuhrman "lying, perjuring, genocidal racist')(09/28/95)
(9) Million Man March in Washington D.C. (10/16/95)
(10)Race flares anew as polarizing issue in U.S. life (10/19/95)

TABLE IV
SAME STORY BUT DIFFERENT THREADS

6) Performance on finding complex story chain: Hurricane Katrina and its effects on government policies is a complex story. Since the start node and end node are not directly related. Table V shows that our story chain algorithm can still find valid and coherent chains for complex stories.

VI. CONCLUSION

In this paper, we proposed a random walk-based finding story chain algorithm. The story chain finding problem is formalized as a divide and conquer bisect search problem.

1) A Blast of Rain but Little Damage as Hurricane Hits South Florida 2005 8 26
3) Hurricane Drenches Florida And Leaves Seven Dead 2005 8 27
4) FEMA, Slow to the Rescue, Now Stumbles in Aid Effort 2005 9 17
5) Millions Are Still Without Power and in Need of Basic Supplies 2005 10 26
6) South Florida Scrambling To Find Emergency Housing 2005 11 11
7) Bush Erred In Responding To Katrina, Lamont Says 2006 8 25
8) Bush failed to act immediately after Hurricane Katrina to waive the requirement that state and local governments match federal rebuilding funds 2007 2 13
9) Bush Consoles Victims of Tornadoes in Alabama and Georgia 2007 3 04
10)Gulf Hits Snags In Rebuilding Public Works 2007 3 31
11)Attacking Bush, Clinton Urges Government Overhaul 2007 4 14

TABLE V
STORY CHAIN ON HOW HURRICANE KATRINA AFFECTS ON GOVERNMENT POLICIES

Two pruning methods are proposed to eliminate redundant articles on the story chain and improve algorithm efficiency: (1) prune least relevant articles; (2) prune redundant articles. The experimental results show that our algorithm can generate coherent story chains with no redundancy and with low computational complexity. Future work includes detect and form story chains with different branches.

REFERENCES

[1] T. H. Haveliwala, "Topic-sensitive pagerank," in *Proceedings of the 11th international conference on World Wide Web*, ser. WWW '02. New York, NY, USA: ACM, 2002, pp. 517–526.

[2] R. Nallapati, A. Feng, F. Peng, and J. Allan, "Event threading within news topics," in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, ser. CIKM '04. New York, NY, USA: ACM, 2004, pp. 446–453.

[3] Q. Mei and C. Zhai, "Discovering evolutionary theme patterns from text: an exploration of temporal text mining," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, ser. KDD '05. New York, NY, USA: ACM, 2005, pp. 198–207.

[4] D. Shahaf and C. Guestrin, "Connecting the dots between news articles," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2010, pp. 623–632.

[5] G. P. C. Fung, J. X. Yu, H. Liu, and P. S. Yu, "Time-dependent event hierarchy construction," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 300–309.

[6] H. Chen and S. Dumais, "Bringing order to the web: automatically categorizing search results," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, ser. CHI '00. New York, NY, USA: ACM, 2000, pp. 145–152.

[7] H. L. Chieu and Y. K. Lee, "Query based event extraction along a timeline," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 2004, pp. 425–432.

[8] F.-r. Lin and C.-H. Liang, "Storyline-based summarization for news topic retrospection," *Decis. Support Syst.*, vol. 45, pp. 473–490, June 2008.

[9] R. Yan, X. Wan, J. Otterbacher, L. Kong, X. Li, and Y. Zhang, "Evolutionary timeline summarization: a balanced optimization framework via iterative substitution," in *Proceedings of the 34th international ACM SIGIR conference*. New York, NY, USA: ACM, 2011, pp. 745–754.

[10] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, "Neighborhood formation and anomaly detection in bipartite graphs," in *In ICDM*, 2005, pp. 418–425.

[11] L. Xiang, Q. Yuan, S. Zhao, L. Chen, X. Zhang, Q. Yang, and J. Sun, "Temporal recommendation on graphs via long- and short-term preference fusion," in *In KDD*. New York, NY, USA: ACM, 2010, pp. 723–732.

[12] R. Angelova and G. Weikum, "Graph-based text classification: learn from your neighbors," in *Proceedings of the 29th annual international ACM SIGIR conference*. New York, USA: ACM, 2006, pp. 485–492.