Inf Syst Front (2014) 16:753–769
DOI 10.1007/s10796-013-9420-2

Finding story chains in newswire articles using random walks

Xianshu Zhu · Tim Oates

Published online: 24 March 2013

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. We further explore how named entities and word relevance can help find relevant news articles and improve algorithm efficiency by creating a co-clustering based correlation graph. Experimental results show that our proposed algorithm can generate coherent story chains without redundancy. The efficiency of the algorithm is significantly improved on the correlation graph.

Keywords Information overload · Random walk · Named entities

1 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 (Haveliwala 2002), 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

This paper is an extended version of our previous paper that was published in the 13th IEEE International Conference on Information Reuse and Integration, 2012.

X. Zhu (✉) · T. Oates
CSEE Department, University of Maryland,
Baltimore County, 1000 Hilltop Circle,
Baltimore MD, USA
e-mail: xianshu1@umbc.edu

T. Oates
e-mail: oates@cs.umbc.edu

coherent story chains, rather than loosely connected pieces of information. Previous research in event threading and tracking (Nallapati et al. 2004; Mei and Zhai 2005) 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 and Guestrin (2010) were the first to address the output coherence problem. However, the algorithm discussed in Shahaf and Guestrin (2010) can be further improved in many ways. Another limitation of unstructured search results is: (2) **Hard to find hidden relationships between two events**: The connections 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 and Guestrin (2010) 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 2 describes related work. Section 3 describes the story chain problem. Section 4 contains a detailed description of our algorithm for finding story chains. Section 5 describes how co-clustering and named entities can help on finding story chains. Section 6 provides experimental results, and Section 7 concludes the paper.

2 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 and Guestrin (2010) 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 previous work (IEEE 13th International Conference on Information Reuse & Integration 2012), we proposed a random walk based method that can handle redundancy and is more computationally efficient. This work significantly improves the efficiency through a co-clustering based correlation graph. Moreover, we capture word relationship and word importance through document-word co-clustering and named entities which can further help to find relevant story chains.

Random walks on graphs to find associations Significant work has been done on random walks on graphs to find associations between two objects. Sun et al. (2005) propose a model based on random walks on bipartite graphs to detect neighborhoods and anomalies. Xiang et al. (2010) propose a temporal personalized random walk method to capture user's temporal preference. Angelova et al. (2006) 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.

Event threading To cope with the information overload problem, many researchers have been working on automatic techniques to organize news stories in a way that users can better understand and analyze them quickly. Event threading is a popular research topic that has been gaining attention for years. Nallapati et al. (2004) cluster news stories into unique events using hierarchical clustering techniques. Each cluster represents a single event. Event threading is done by capturing similarity dependencies among clusters. If similarity between two clusters is high, then a dependency link should be added between these two clusters. There are some drawbacks of this method. First, clustering is just a coarse-grained technique to segment news stories into events. It is unknown whether clustering can achieve good results on news stories that contain events with very similar features. Even though it can cluster news articles into separate events, each event can be characterized from several aspects and events will have different relational structures under different context. Events dependencies cannot be simply modeled by clusters similarities. For example, the O.J. Simpson trial contains many events including the low-speed chase, arrest, evidence collection (DNA evidence, glove), and racial differences. Most events in this story contain keywords like "Simpson", "Ronald Goldman", "Nicole Brown Simpson", "murder", "blood". It is not easy to separate these events into clusters. Moreover, events like "The discovery of the blood marks and leather glove that Mark Fuhrman found on Simpson's property hours after the murders took place." can be considered as evidence in the trial and can also be considered as an event related to racial issues. In our method, we explore the relationships between clusters by emphasizing named entities.

Recently, various approaches (Mei and Zhai 2005; Perkio et al. 2004; Spiliopoulou et al. 2006; Jo et al. 2011; He et al. 2009; Morinaga and Yamanishi 2004) have been proposed to model topic temporal evolution. Mei et al. (2005) study the problem of detecting topics from temporal text streams. They perform theme level word clustering instead of document level clustering. A topic evolution graph is built and used to trace topic transitions. Perkio et al. (2004) extract temporal topic patterns using a multinomial PCA model. MONIC in Spiliopoulou et al. (2006) is a framework for monitoring and tracking cluster transitions over time. Jo et al. (2011) study how to discover the evolution of topics over time in a time-stamped document collection with emphasis on revealing the topology of topic evolution. A major difference between our work and their's is that we do not exhaustively extract the whole story evolution topology, but find specific story evolution chains depending on users' queries.

Fung et al. (2007) 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. Chen et al. (2000) 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 Timeline generation (Chieu and Lee 2004; Lin and Liang 2008; Yan et al. 2011) generates news story evolution tracjectories along a timeline given query related news collections. Yan et al. (2011) 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.

Using named entities for event threading Named entities are important features to characterize news events. An event usually can be described as *when*, *where*, *who* and *what* happened. The corresponding named entities are *Time*, *Location*, *People*, *Event*. Makkonen et al. (2002, 2004) split the term space into groups of terms that have similar meaning: location, names, and temporal expressions. A similarity metric is applied group-wise, for example, to compare names in one document with names in another. Different similarity measures are combined via a weighted sum to obtain confidence scores for each story. The weight function is learned using a perceptron (Makkonen et al. 2004). Kumaran et al. (2004) split the terms into all terms in the document, only the named entities, and only the non-named entity terms. They find that addressing named entities preferentially is useful only in certain situations. Thus they proposed a multi-stage new event detection system. Ahmed et al. (2009) detect and track terrorism news threads by extracting event signatures (people involved, organizations mentioned, locations, dates, event describing phrases, etc.). In our work, named entities are considered as of higher importance than other non-named entities in the story chain finding process.

3 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 Fig. 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 Fig. 1a 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 Fig. 1b 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 along which 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.

4 Finding story chains through random walks on graphs

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 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 the start article and t is the 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.

The advantages of this algorithm are:

- Coherence and relevance: This method can create high quality chains in terms of coherence and relevance. See Section 4.2 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 Shahaf and Guestrin (2010). 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.

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 $C = s - a_1 - a_2 - \cdots - a_i - t$

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

4.1 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 the word 'hurricane'. However these two articles might be related.

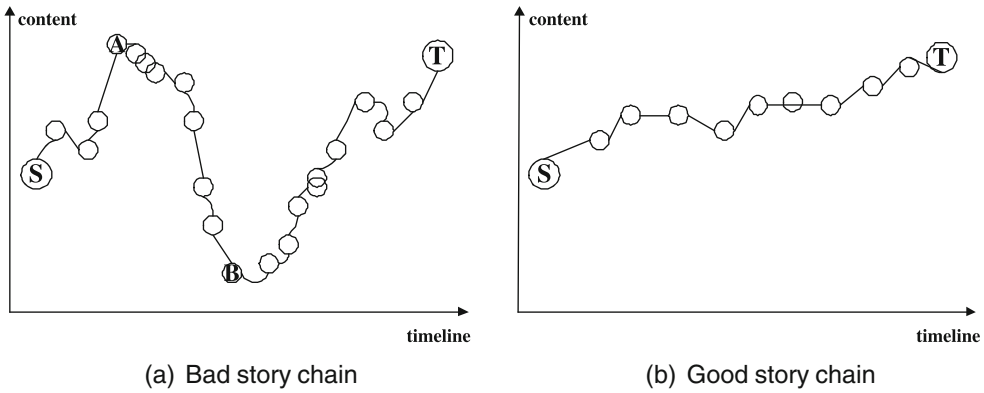

Fig. 1 Examples of good story chain vs. bad story chain

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

We construct a bipartite graph, as shown in Fig. 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 as shown in Fig. 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 Fig. 2.

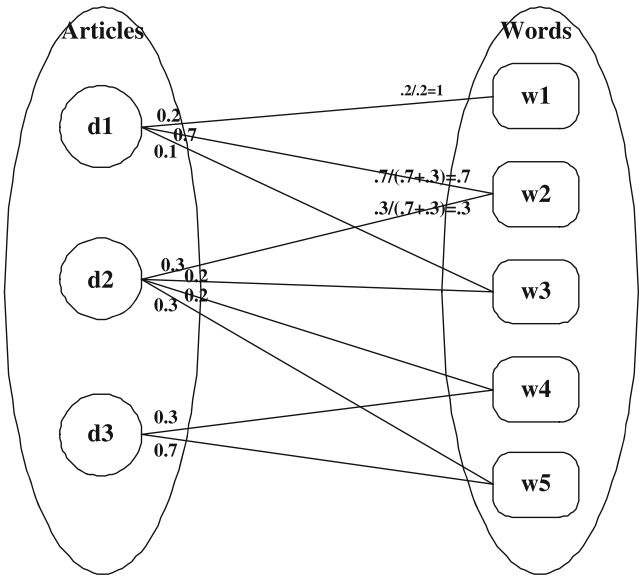

Fig. 2 Compute link strength (Link strength is computed based on random walks on a bipartite graph. The link strength between documents d_1 and d_2 is high, if random walks start from d_1 can frequently reach d_2)

4.2 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 first compute $\operatorname{argmax}_{d_i} r_s(d_i) * r_t(d_i)$, 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 generates a coherent and relevant story chain.

4.3 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.

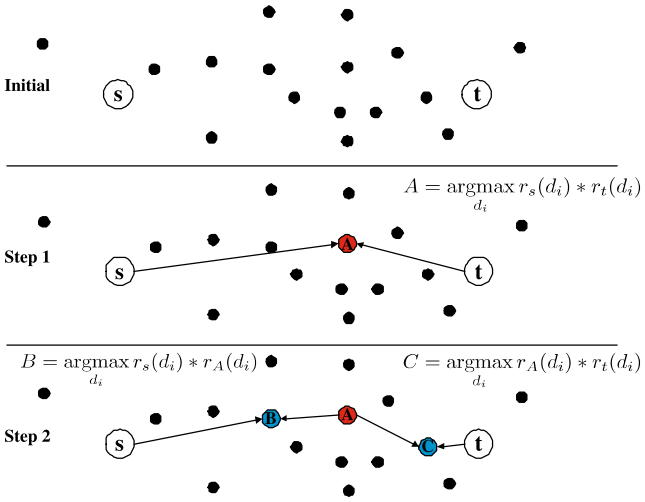

Fig. 3 Illustration of search algorithm (The story chain finding problem is formalized as a bisecting search problem)

4.3.1 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 Fig. 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

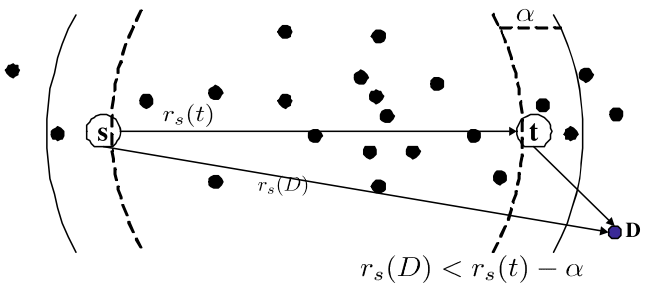

Fig. 4 Prune the least relevant articles (r_s are probability values). The larger the r_s value, the closer the relationship between two nodes will be

$r_s(d_i)$ 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 scores 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 $r_s(d_i) < r_s(t)$.

4.3.2 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 Fig. 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.

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

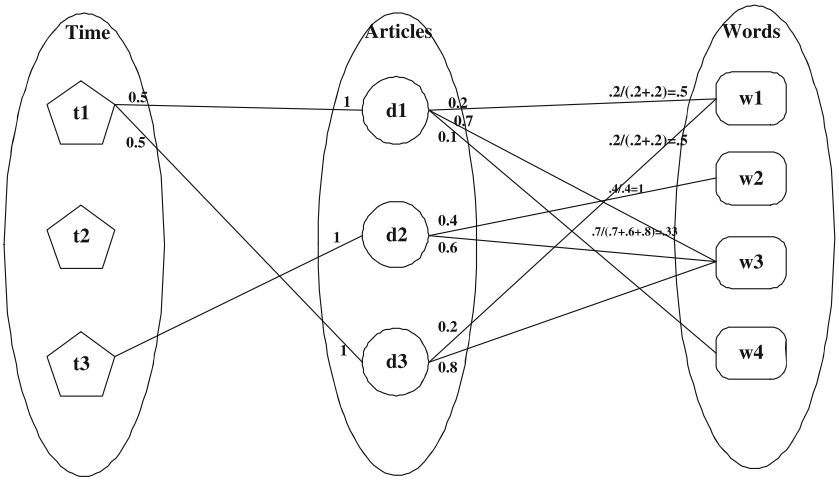

Fig. 5 Method to solve redundancy problem (Random walks on a tripartite graph is used to find documents that are similar both in content and time)

time decay. We assume the random walk starts from d_a. The extended random walk formula is:

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

where M is the transition matrix for the tripartite graph. β is the damping factor which represents the chance that the random walk stops follow the edges in the tripartite graph and jump to another random node. $\mathbf{N}$ is a vector indicating which nodes the random walk will jump to after a restart.

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

The vector $\mathbf{N}$ 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. The reason why we choose an exponential decay function is that as the publication date of two documents differs more significantly, the probability that the two documents are talking about the same event will decrease exponentially.

5 Making the random walk more directional through co-clustering and named entities

In this section, we further explore ways to improve story chain finding process.

5.1 Propagate the relevance of word nodes to document nodes through a co-clustering based correlation graph

The document and word bipartite graph does not consider the relationships among a single type of node. However, words are not independent. Some words are semantically close and some are not related. The random walk approach can help find relationships between documents that don't have overlapped words. However, the pure random walk may bring up some irrelevant documents. In Fig. 6, articles $d1$ and $d2$ are related. They are all about the Hurricane Katrina natural disaster. Article $d2$ talks about how the government reacted to Hurricane Katrina. $d3$ is purely about government policy on some unrelated issues. $d1$ and $d3$ may or may not be correlated. In this example, they are most probably not related. However, a pure random walk starting from $d1$ will treat $d3$ as a related article. The question is can we propagate the relevance of one type of node

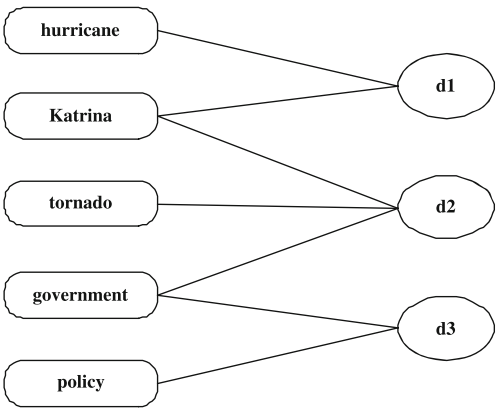

Fig. 6 Single type of nodes are independent in bipartite graph (There are no edges among word/document nodes)

to another type of node so as to make the random walk more directional? More specifically, can we find relationships among word nodes and propagate these relationships to the document nodes?

Document and word co-clustering can help with such relationship propagation. Instead of working directly on a word-document bipartite graph where words are treated as independent nodes, we work on document and word co-clusters which group semantically close words together and propagate the relevance of word nodes to document nodes. Document and word co-clusters are generated using an information theoretic co-clustering algorithm (Dhillon et al. 2003). The information-Theoretic Co-clustering algorithm (Dhillon et al. 2003) is a widely used algorithm with solid mathematical foundation. The co-clustering problem (see Fig. 7) is defined as finding a pair of maps from rows to row-clusters and from columns to column-clusters with minimum mutual information loss:

$$I(X;Y) - I(\hat{X};\hat{Y}) = KL(p(x,y)\|q(x,y)) \tag{3}$$

where $p(x,y)$ is the input distribution and $q(x,y) = p(\hat{x},\hat{y})p(x|\hat{x})p(y|\hat{y})$. The loss in mutual information can be expressed as a weighted sum of relative entropies between row distribution $p(Y|x)$ and row cluster distribution $q(Y|\hat{x})$:

$$KL(p(x,y)\|q(x,y)) = \sum_{\hat{x}} \sum_{x\in\hat{x}} p(x) KL(p(Y|x)\|q(Y|\hat{x})) \tag{4}$$

and can also be expressed as a weighted sum of relative entropies between column distribution $p(X|y)$ and column cluster distribution $a(X|\hat{y})$:

$$KL(p(x,y)\|q(x,y)) = \sum_{\hat{y}} \sum_{y\in\hat{y}} p(y) KL(p(X|y)\|q(X|\hat{y})) \tag{5}$$

where $q(Y|\hat{x})$ or $q(X|\hat{y})$ can be viewed as a row or column cluster prototype "centroid" in the k-means algorithm. The basic idea is similar to the k-means clustering algorithm. However, it uses a different formula to compute the cluster "prototype", which is the centroid in the k-means algorithm.

Then we construct a correlation graph that represents the relationship between words, word clusters, document clusters and documents as shown in Fig. 8.

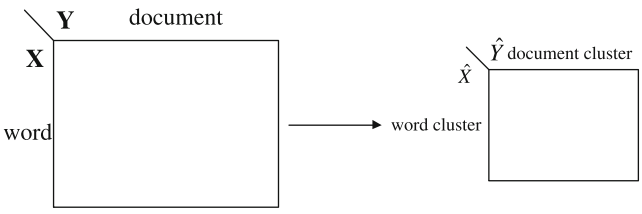

Fig. 7 Information-theoretic co-clustering

5.1.1 Random walk on the correlation graph

Random walks on the correlation graph can be used to find document relevance scores. Random walks on this correlation graph match natural ways of human thinking. Given a start document as a hint, a natural way of looking for relevant documents is to first look at ones that lie in the same document cluster. And then looking for documents in other clusters that are most relevant to its own cluster. If the goal is to look for relevant documents that contain more new information or concepts, people would prefer to search out of its own cluster. This search pattern can be easily achieved by adjusting the λ value in the transition matrix which will be described in detail in Section 5.1.2 below.

5.1.2 Correlation graph construction

As shown in Fig. 8, we construct a correlation graph $G = \langle V_W \cup V_{WC} \cup V_{DC} \cup V_D, E\rangle$, where word node $V_W = \{w_i|1 \le i \le |W|\}$, document node $V_D = \{d_i|1 \le i \le |D|\}$, word cluster node $V_{WC} = \{wc_i|1 \le i \le |WC|\}$ and document cluster node $V_{DC} = \{dc_i|1 \le i \le |DC|\}$. As a random walk process will stay in the right part of the correlation graph inside the rectangle, we only compute edge weights between word clusters and document clusters, and between document clusters and documents.

Edge weight between word clusters and document clusters The TFIDF weighting scheme is a common method in calculating a document's relevance to a word. In other words, $tf\text{-}idf_{w|d}$ assigns to word w a weight in document d. Document cluster j's relevance to word cluster i can be calculated

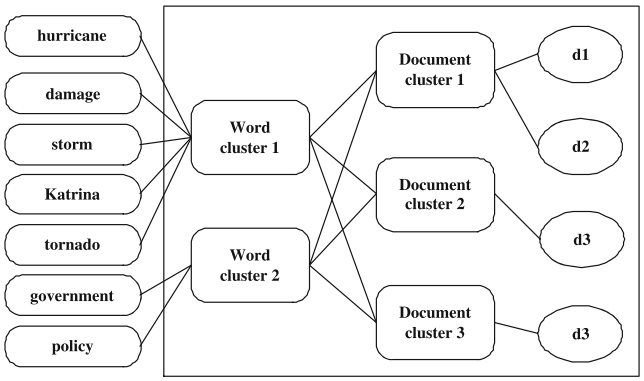

Fig. 8 Correlation graph used for finding story chain

as the sum of the document cluster's relevance to each of the words in the word cluster, which is:

$$\begin{aligned} &RelevanceScore(WC_i|DC_j) \\ &= \sum_{k=1}^{|WC_i|} RelevanceScore(w_k|DC_j) \\ &= \sum_{k=1}^{|WC_i|}\sum_{m=1}^{|DC_j|} RelevanceScore(w_k|d_m) \\ &= \sum_{k=1}^{|WC_i|}\sum_{m=1}^{|DC_j|} tf\text{-}idf(w_k|d_m) \end{aligned} \tag{6}$$

where WC_i denotes word cluster i, DC_j denotes document cluster j, w denotes a word, d denotes a document, $|WC_i|$ denotes the number of words in the word cluster, and $|DC_j|$ denotes the number of documents in the document cluster.

Edge weight between documents and document clusters Similarly, document cluster j's relevance to document i can be calculated as the sum of the document cluster's relevance to each of the words in the document, which is:

$$\begin{aligned} &RelevanceScore(d_i|DC_j) \\ &= \sum_{k=1}^{|d_i|} tf(w_k|d_i) RelevanceScore(w_k|DC_j) \\ &= \sum_{k=1}^{|d_i|} tf(w_k|d_i) \sum_{m=1}^{|DC_j|} RelevanceScore(w_k|d_m) \\ &= \sum_{k=1}^{|d_i|} tf(w_k|d_i) \sum_{m=1}^{|DC_j|} tf\text{-}idf(w_k|d_m) \end{aligned} \tag{7}$$

After computing the edge weights, we can construct the transition matrix M, such that every column sums to 1:

$$M = \begin{pmatrix} 0 & \lambda M_{dc\text{-}wc} & 0 \\ M_{wc\text{-}dc} & 0 & M_{d\text{-}dc} \\ 0 & (1-\lambda) M_{dc\text{-}d} & 0 \end{pmatrix} \tag{8}$$

where $M_{dc\text{-}wc}$ stores the transition probability from document clusters to word clusters, $M_{wc\text{-}dc}$ stores the transition probability from word clusters to document clusters, $M_{d\text{-}dc}$ stores the transition probability from documents to document clusters, and $M_{dc\text{-}d}$ stores the transition probability from document clusters to documents. Every matrix is normalized to make each column sum to 1. $M_{dc\text{-}wc}$ and $M_{dc\text{-}d}$ are multiplied by a parameter λ and $1-\lambda$ to further make the transition matrix M column sum up to 1. The parameter λ helps the document cluster node to make a hierarchial decision of which way to go. More specifically, a document cluster node first chooses to jump to word cluster nodes with probability λ and go to document nodes with probability $1-\lambda$. Then, it follows the probability value in $M_{dc\text{-}wc}$ or $M_{dc\text{-}d}$ to choose which word cluster or document to jump to.

5.2 Emphasize named entities

TFIDF weighting approach assumes that the words in a document are considered as unordered and independent elements. This approach ignores the fact that context information can also affect word importance. For example, words appearing in the headline are more important than words appearing in the body. Named entities, such as people, locations and organizations, are more useful and important for representing the main topic of the content. We further adaptively adjust the edge weights on the correlation graph to give more attention to named entities in different start nodes. In this way, named entities are considered as a key factor to direct the story chain finding process. In this paper, we use OpenCalais (http://www.opencalais.com/) named entity recognition tool to extract named entities.

5.2.1 Graph weights biased on named entities

As shown in Eq. 7, the transition probability from a document cluster to documents depends on aggregation of the document words' relevance to the document cluster. It is independent of the random walk start node. However, our motivation for using a random walk is to find document relevance scores to a given document node but not to a document cluster. In the event threading literature, a document cluster is commonly treated as a single event (Nallapati et al. 2004). Any document in that document cluster can well represent the event. This is not exactly true, since an event can be characterized in several ways. For example, when talking about the O.J. Simpson trial, we may view it from variety of aspects, such as criminal evidence, racial issue, detectives, etc. Thus, it is necessary to adjust the edge weights to bias more on words in the start or end node. Moreover, named entities such as persons, locations, organizations and historical events have been shown useful for capturing news event features. We adjust edge weights to bias more toward named entities in random walk start and end nodes. More detail follows:

1. Create named entities co-occurrence matrix CM_i for each document cluster i. The co-occurrence frequency of two named entities $co\text{-}occur(NE_m, NE_n)$ is calculated as the co-occurrence rate in a window size of document length. The co-occurrence matrix is defined as the aggregation of co-occurrence rates for

all documents in the document cluster normalized by the number of documents in the document cluster, see Eq. 9. The co-occurrence matrix is normalized to column sum to 1.

$$CM_i(m,n) = \frac{\sum_{k=1}^{|DC_i|} co\text{-}occur(NE_m, NE_n)_k}{|DC_i|} \quad (9)$$

2. Compute named entity weight vector $W = CM_i * C_d$, where C_d is the named entity count vector for document d. Each element i in $C_d(i)$ is defined as

$$C_d(i) = \frac{\text{number of named entity } i \text{ appeared in document } d}{\text{Total number of named entities in document d}} \quad (10)$$

In the computed weight vector W, the named entities that appeared in d are given the highest weight. The named entities that do not appear in d but are closely related to those that do appear are given less weight. All other named entities are assigned the lowest weights.

3. Compute relevance score with respect to named entities for all documents in the dataset. The relevance score for document d_i is defined as:

$$RelevanceScoreNE(d_i) = W^T \cdot C_{d_i} \quad (11)$$

4. Adjust document cluster to document edge weights by parameter β, where $\beta \in [0, 1]$:

$$RelevanceScoreUpdate(d_i|DC_j) = \beta * RelevanceScore(d_i|DC_j) + (1-\beta) * RelevanceScoreNE(d_i) \quad (12)$$

5. Adjust document cluster to word cluster edge weights by parameter γ, where $\gamma \in [0, 1]$:

$$RelevanceScoreUpdate(WC_i|DC_j) = \gamma * RelevanceScore(WC_i|DC_j) + (1-\gamma) * RelevanceScoreNE(WC_i) \quad (13)$$

Note that the adjusted edge weights are always influenced by named entities appearing in the random walk start node but not the story chain start node. By adjusting edge weights, document clusters can emphasize different named entities.

6 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 show how different pruning methods affect story chain construction and the efficiency of the algorithm. We also show how document-word co-clustering and named entities 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, so we constructed our own data sets on real news articles which will be discussed in Section 6.1 in detail.

- **Connecting-Dots** (Shahaf and Guestrin 2010): 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 on 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)** (Nallapati et al. 2004): 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 6.3.2 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 ranked the story chains from best (5) to worst (1) by relevance, coherence, coverage and redundancy. For example, we define no redundancy as very good (5), less than half of the news articles are redundant as good (4), half of the news articles are redundant as fair (3), more than half are redundant as not good (2) and all are redundant as bad (1).

6.1 Data set

We constructed data sets for three news topics: the O.J. Simpson Trial, Hurricane Katrina and the earthquake and tsunami in Japan in 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

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

Table 1 Data Set Information

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

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 in 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 the 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 1 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.

6.2 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 2. 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 to each other. For example, story C1 is a complex story since Hurricane 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.

6.3 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). We further explore word relationships through random walks on a co-clustering based correlation graph. We use information-theoretic co-clustering software (http://www.cs.utexas.edu/users/dml/Software/cocluster.html) to cluster documents and words into clusters. Named entities are extracted using OpenCalais (http://www.opencalais.com/). We show the performance for random walks on co-clustering based correlation graph (CLS) as well. The performance evaluation results are based on human judegments through MTurk. Story chains are ranked from best (5) to worst (1) by relevance, coherence, coverage and redundancy.

Note that one issue for the co-clustering based algorithm (CLS) is how to determine the number of word and document clusters. In this paper, our focus is to explore whether document and word co-clustering can help on finding relevant documents or not. Thus, we simply specify the number of document and word clusters to be 20 and 23 respectively in the experiments below. In the future, we will try to find better ways to determine what are the optimal number for document and word clusters.

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

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

Table 2 Initial information of the story chains

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

6.3.1 Performance of the random walk based algorithm

- **Performance of PLR (prune irrelevant articles):** 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 the Japan Earthquake in 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 9 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 9 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.
- **Performance of PR (only prune redundant articles):** We show the experimental results for PR. Figure 10 shows that the story chain contains much less redundant articles with PR.
- **Performance of both PLR-PR:** Figure 11 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.

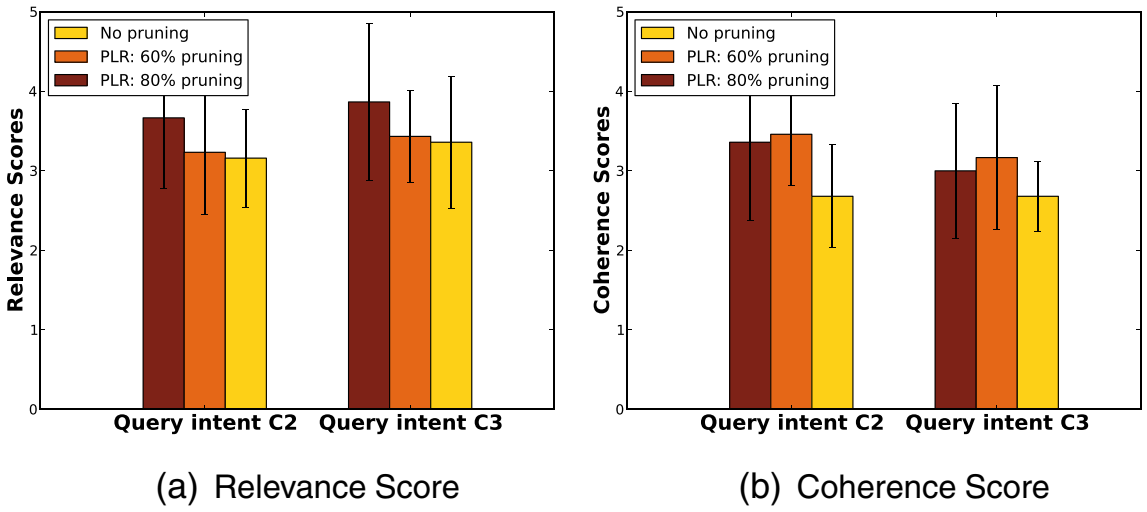

Fig. 9 Performance of PLR

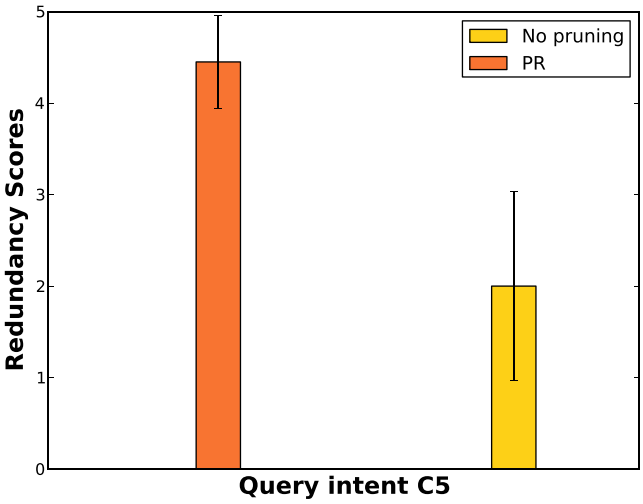

Fig. 10 Performance of PR

6.3.2 Performance of co-clustering based correlation graph (CLS)

Figure 12 compares the performance among four algorithms: Event threading, No-pruning and Both-pruning (PLR-PR) and co-clustering based (CLS). 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. The CLS method achieves almost the same performance as the Both-pruning(PLR-PR). The CLS method slightly outperforms the PLR-PR method in terms

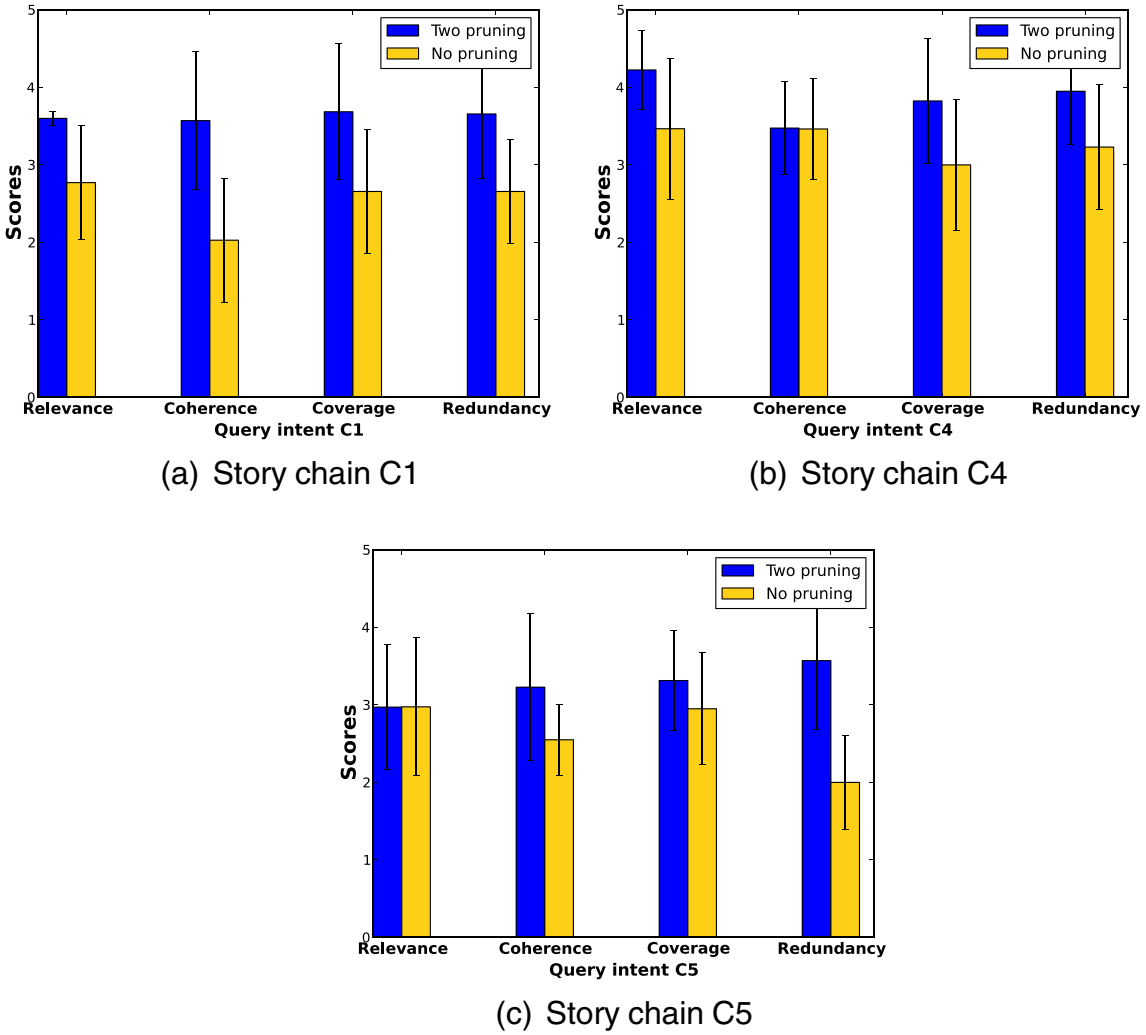

Fig. 11 Performance of PLR-PR and No-pruning

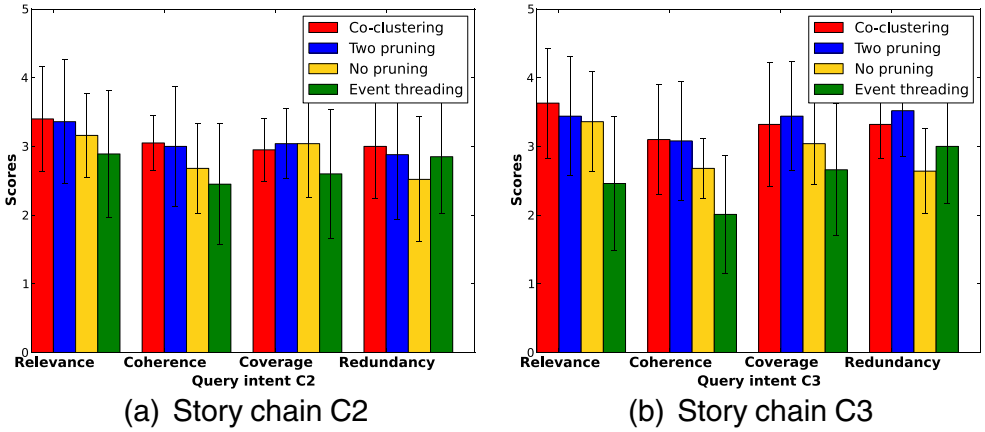

Fig. 12 Performance of PLR-PR, No-pruning and event threading

(a) Story chain C2 (b) Story chain C3

of relevance which shows that co-clustering and named entities can help on finding relevant documents.

6.3.3 Computational complexity

The computation overhead of the story chain algorithm PR-PLR 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 (Shahaf and Guestrin 2010), 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 methods, which further speeds up the random walk computation. Figure 13 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 the very beginning. The computation overhead is mainly determined by the size of the graph, and the correlation graph size is significantly reduced from $|D| + |W|$ (document-word bipartite graph) to $n + |D|$, in which n is the total number of document and word clusters. Therefore, the co-clustering based story chain algorithm will introduce an even lower computation overhead.

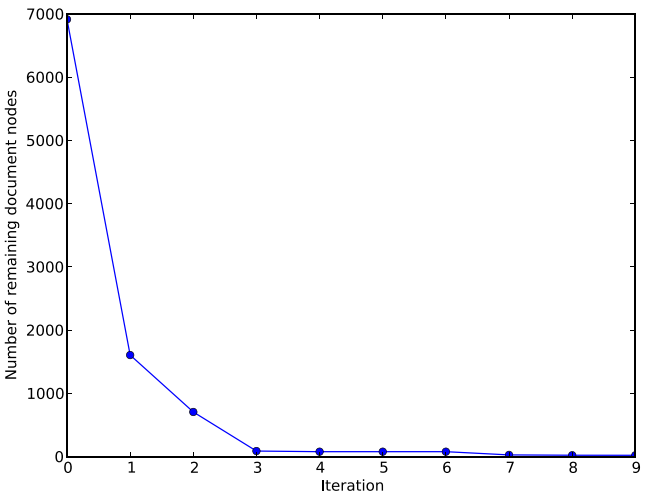

Fig. 13 Number of remaining documents in each iteration

Table 3 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, PLR-PR and CLS 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

Table 3 Running time of the algorithm

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
CLS	0.03	0.05	0.06	0.1	0.13

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 CLS algorithm is much more faster than PLR-PR which only requires less than 10% of what the no-pruning method requires. This is because the graph size of the correlation graph is much more smaller than document-word bipartite graph. The algorithm is running on a laptop with Intel Core i5 2.67 GHz and 4G memory. The average running time for the no-pruning algorithm is 5 min and the algorithm with both pruning algorithm reduces the time to only 2 min. 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.

Table 4 Same story but different threads (PLR-PR)

(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 5 Story chain on how Hurricane Katrina affects on government policies

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

(b) Random walk on co-clustering based correlation graph (CLS)
- (1) A Blast of Rain but Little Damage as Hurricane Hits South Florida 2005 8 26
- (2) Hurricane Slams Into Gulf Coast; Flooding Spreads 2005 9 25
- (3) Politics Stalls Plan to Bolster Flood Coverage 2006 5 15
- (4) Small Clause, Big Problem; A Detail in Insurance Policies Is an Issue in Hurricane Claims 2006 8 4
- (5) A Bank Survives Katrina. Now, the Hard Part. 2006 9 3
- (6) A Proposal for Federal Protection From Catastrophe Divides Insurers 2006 10 31
- (7) Louisiana Disputes a Bill From FEMA for Hurricane Aid 2006 12 7
- (8) Operating Costs Are Imperiling Disaster Fund 2007 1 18
- (9) In Washington, Contractors Take On Biggest Role Ever 2007 2 4
- (10) The recovery is going too slowly, and government should take to speed the process 2007 2 13
- (11) Sense of Progress on Disaster Preparedness 2007 2 20
- (12) Gulf Hits Snags In Rebuilding Public Works 2007 3 31
- (13) Ex-Aide Details A Loss of Faith In the President 2007 4 1
- (14) HUD Institutes Changes To Speed Storm Repairs 2007 4 7 (Louisiana homeowners may get faster access to rebuilding grants after a federal decision late last month that is forcing the state to change its slow-moving 7.5-billion "Road Home" program to repair houses damaged or destroyed by hurricanes in 2005.)
- (15) Attacking Bush, Clinton Urges Government Overhaul 2007 4 14

6.3.4 Same story, different thread

In this experiment (Table 4), 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 (PLR-PR) is quite adaptive to users' queries. It helps web users to easily understand different aspects of a story.

6.3.5 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 5 shows that our story chain algorithms PLR-PR and CLS can still find valid and coherent chains for complex stories. News article (8) in story chain using PLR-PR talks about Bush consoles Victims of Tornadoes in Alabama and Georgia which is not related to both government policies and hurricane Katrina. News articles in story chain using CLS are more relevant to how government policies play a role in Hurricane Katrina aftermath.

7 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. 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. We further explore how document-word co-clustering and named entities can help find document relationships. The experimental results show that our algorithm can generate coherent story chains with no redundancy and with low computational complexity. Random walks on a co-clustering based correlation graph can largely improve efficiency of the algorithm while producing good story chains. Future work includes detect and form story chains with different branches. Generate more datasets and test our algorithm on the new datasets.

References

Ahmed, S.T., Bhindwale, R., Davulcu, H. (2009). Tracking terrorism news threads by extracting eventsignatures. In *Proceedings of the 2009 IEEE international conference on intelligence and security informatics, ISI'09* (pp. 182–184). Piscataway: IEEE Press.

Angelova, R., & Weikum, G. (2006). Graph-based text classification: learn from your neighbors. In *Proceedings of the 29th annual international ACM SIGIR conference* (pp. 485–492). New York: ACM.

Chen, H., & Dumais, S. (2000). Bringing order to the web: automatically categorizing search results. In *Proceedings of the SIGCHI conference on human factors in computing systems, CHI '00* (pp. 145–152). New York: ACM.

Chieu, H.L., & Lee, Y.K. (2004). Query based event extraction along a timeline. In *Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval* (pp. 425–432). New York: ACM.

Dhillon, I.S., Mallela, S., Modha, D.S. (2003). Information-theoretic co-clustering. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining, KDD '03* (pp. 89–98). New York: ACM.

Fung, G.P.C., Yu, J.X., Liu, H., Yu, P.S. (2007). Time-dependent event hierarchy construction. In *Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '07* (pp. 300–309). New York: ACM.

Haveliwala, T.H. (2002). Topic-sensitive pagerank. In *Proceedings of the 11th international conference on world wide web, WWW '02* (pp. 517–526). New York: ACM.

He, Q., Chen, B., Pei, J., Qiu, B., Mitra, P., Giles, L. (2009). Detecting topic evolution in scientific literature: How can citations help? In *Proceeding of the 18th ACM conference on Information and knowledge management, CIKM '09* (pp. 957–966). New York: ACM.

Jo, Y., Hopcroft, J.E., Lagoze, C. (2011). The web of topics: discovering the topology of topic evolution in a corpus. In *Proceedings of the 20th international conference on world wide web, WWW '11* (pp. 257–266). New York: ACM.

Kumaran, G., & Allan, J. (2004). Text classification and named entities for new event detection. In *Proceedings of the 27th annual international ACM SIGIR conference on research and development in information, retrieval, SIGIR '04* (pp. 297–304). New York: ACM.

Lin, F.-r., & Liang, C.-H. (2008). Storyline-based summarization for news topic retrospection. *Decision Support Systems*, *45*, 473–490.

Makkonen, J., Ahonen-Myka, H., Salmenkivi, M. (2002). Applying semantic classes in event detection and tracking. In *Proceedings of international conference on natural language process* (pp. 175–183). Mumbai: Springer.

Makkonen, J., Ahonen-Myka, H., Salmenkivi, M. (2004). Simple semantics in topic detection and tracking. *Information Retrieval*, *7*, 347–368.

Mei, Q., & Zhai, C. (2005). 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, KDD '05* (pp. 198–207). New York: ACM.

Morinaga, S., & Yamanishi, K. (2004). Tracking dynamics of topic trends using a finite mixture model. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining, KDD '04* (pp. 811–816). New York: ACM.

Nallapati, R., Feng, A., Peng, F., Allan, J. (2004). Event threading within news topics. In *Proceedings of the thirteenth ACM international conference on information and knowledge management, CIKM '04* (pp. 446–453). New York: ACM.

Perkio, J., Buntine, W., Perttu, S. (2004). Exploring independent trends in a topic-based search engine. In *Proceedings of the 2004 IEEE/WIC/ACM international conference on web intelligence, WI '04* (pp. 664–668). Washington: IEEE Computer Society.

Shahaf, D., & Guestrin, C. (2010). Connecting the dots between news articles. In *Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '10* (pp. 623–632). New York: ACM.

Spiliopoulou, M., Ntoutsi, I., Theodoridis, Y., Schult, R. (2006). Monic: modeling and monitoring cluster transitions. In *Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '06* (pp. 706–711). New York: ACM.

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

Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., Sun, J. (2010). Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '10* (pp. 723–732). New York: ACM.

Yan, R., Wan, X., Otterbacher, J., Kong, L., Li, X., Zhang, Y. (2011). Evolutionary timeline summarization: A balanced optimization framework via iterative substitution. In *Proceedings of the 34th international ACM SIGIR conference* (pp. 745–754). New York: ACM.

Zhu, X., Oates, T. (2012). Finding story chains in newswire articles. In *IEEE 13th International conference on information reuse & integration, IRI 2012, Las Vegas, NV, USA, August 8–10*. IEEE.

Xianshu Zhu is currently a Ph.D candidate in Computer Science at University of Maryland Baltimore County (UMBC). She received her B.S. and M.E. in Computer Science both from Hunan University, Changsha, Hunan, China in June 2004 and June 2007 respectively. She has been actively engaged in research on text mining, information retrieval and natural language processing, especially extracting structured information from unstructured news text and presenting them in a more semantically meaningful way by creating story maps and finding story chains so as to help people get a big picture of the news story quickly and improve their news browsing experience.

Tim Oates is a Professor in the Department of Computer Science and Electrical Engineering at the University of Maryland Baltimore County. He received B.S. degrees in Computer Science and Electrical Engineering from North Carolina State University in 1989, and M.S. and Ph.D. degrees from the University of Massachusetts Amherst in 1997 and 2000, respectively. Prior to coming to UMBC in the Fall of 2001, spent a year as a postdoc in the Artificial Intelligence Lab at the Massachusetts Institute of Technology. In 2004 Dr. Oates won an NSF CAREER award. He is an author or co-author of more than 100 peer reviewed papers and is a member of the Association for Computing Machinery and the Association for the Advancement of Artificial Intelligence.