Connecting the Dots Between News Articles

Dafna Shahaf
Carnegie Mellon University
dshahaf@cs.cmu.edu

Carlos Guestrin
Carnegie Mellon University
guestrin@cs.cmu.edu

ABSTRACT

The process of extracting useful knowledge from large datasets has become one of the most pressing problems in today's society. The problem spans entire sectors, from scientists to intelligence analysts and web users, all of whom are constantly struggling to keep up with the larger and larger amounts of content published every day. With this much data, it is often easy to miss the big picture.

In this paper, we investigate methods for automatically connecting the dots – providing a structured, easy way to navigate within a new topic and discover hidden connections. We focus on the news domain: given two news articles, our system automatically finds a coherent chain linking them together. For example, it can recover the chain of events starting with the decline of home prices (January 2007), and ending with the ongoing health-care debate.

We formalize the characteristics of a good chain and provide an efficient algorithm (with theoretical guarantees) to connect two fixed endpoints. We incorporate user feedback into our framework, allowing the stories to be refined and personalized. Finally, we evaluate our algorithm over real news data. Our user studies demonstrate the algorithm's effectiveness in helping users understanding the news.

1. INTRODUCTION

"Can't Grasp Credit Crisis? Join the Club", stated David Leonhardt's article in the New York Times. Credit crisis had been going on for seven months by that time, and had been extensively covered by every major media outlet throughout the world. Yet many people felt as if they did not understand what it was about.

Paradoxically, the extensive media coverage might have been a part of the problem. This is another instance of the *information overload* problem, long recognized in the computing industry. Users are constantly struggling to keep up with the larger and larger amounts of content that is being published every day; with this much data, it is often easy to miss the big picture.

For this reason, there is an increasing need for techniques to present data in a meaningful and effective manner. In this paper, we investigate methods for automatically *connecting the dots* – providing a structured, easy way to uncover hidden connections between two pieces of information. We believe that the ability to connect dots and form a logical, coherent story lies at the basis of understanding a topic.

We focus on the news domain: given two news articles, our system automatically finds a coherent *story* (chain of articles) linking them together. For example, imagine a user who is interested in the financial crisis and its effect on the health-care reform. The user vaguely recalls that the financial crisis is related to the decline of home prices in 2007. The user would then choose representative articles for those two topics and feed them to our system. An output chain may look like this (parenthesized text not part of output):

1.3.07	**Home Prices Fall** Just a Bit
3.4.07	Keeping Borrowers Afloat
	(Increasing delinquent mortgages)
3.5.07	A **Mortgage Crisis** Begins to Spiral, ...
8.10.07	... **Investors Grow Wary** of Bank's Reliance on Debt.
	(Banks' equity diminishes)
9.26.08	Markets **Can't Wait for Congress** to Act
10.4.08	**Bailout Plan** Wins Approval
1.20.09	Obama's Bailout Plan **Moving Forward**
	(... and its effect on health benefits)
9.1.09	Do Bank Bailouts **Hurt Obama on Health?**
	(Bailout handling can undermine health-care reform)
9.22.09	Yes to Health-Care Reform, but Is This the Right Plan?

The chain mentions some of the key events connecting the mortgage crisis to healthcare, including the bailout plan. Most importantly, the chain should be *coherent*: after reading it, the user should gain a better understanding of the progression of the story.

To the best of our knowledge, the problem of connecting the dots is novel. Previous research (e.g., [19, 13, 18, 17, 4, 6]) focused on organizing news articles into hierarchies or graphs, but did not address the notion of output coherence.

Our main contributions are

- Formalizing characteristics of a good story and the notion of *coherence*.
- Formalizing *influence* with no link structure.
- Providing an efficient algorithm for connecting two fixed endpoints while maximizing chain coherence (with theoretical guarantees).
- Incorporating feedback and interaction mechanisms into our system, tailoring stories to user preferences.
- Evaluating our algorithm over real news data and demonstrating its utility to news-readers via a user study.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD'10, July 25–28, 2010, Washington, DC, USA.
Copyright 2010 ACM 978-1-4503-0055-110/07 ...$10.00.

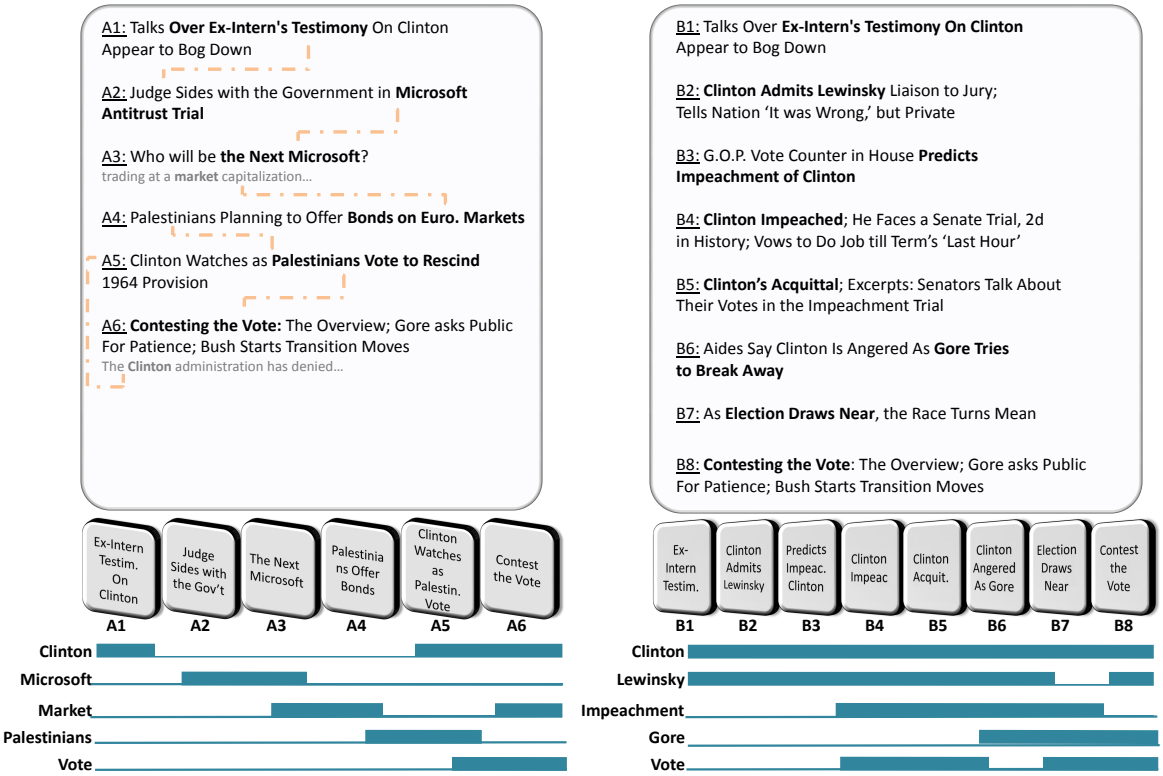

Figure 1: Two examples of stories connecting the same endpoints. Left: chain created by shortest-path (dashed lines indicate similarities between consecutive articles). Right: a more coherent chain. Activation patterns for each chain are shown at the bottom; the bars indicate appearance of words in the article above them.

Our methods are also directly applicable to many other domains. Email, research papers, and military intelligence analysis are but a few of the domains in which it would be immensely useful to discover, extract, and automatically connect the dots.

2. SCORING A CHAIN

2.1 What makes a story good?

Our goal is to find a good path between two articles, s and t. A natural thing to do would be to construct a graph over the articles and find the shortest s-t path. Since there are no edges between articles, we will have to add them ourselves, e.g., by linking similar articles together.

However, this simple method does not necessarily yield a good chain. Suppose we try to find a coherent chain of events between Clinton's alleged affair and the 2000 election Florida recount. We pick two representative documents,

s: *Talks Over Ex-Intern's Testimony On Clinton Appear to Bog Down (Jan 1998)*

t: *Contesting the Vote: The Overview; Gore asks Public For Patience (Nov 2000)*

and find a shortest path between them. The result is shown on Figure 1 (left). This chain of stories is rather erratic, passing through the Microsoft trial, Palestinians, and European markets before returning to Clinton and American politics. Note that each transition, when examined out of context, is reasonable: for example, the first and the second articles are court-related. Those correlations are marked by dashed lines in Figure 1.

The problem seems to lie with the locality of shortest-path. Every two articles along the chain are related, but there is no *global*, coherent theme to the chain as a whole. Rather, shortest-path chains may exhibit *stream-of-consciousness* behaviour, linking s and t by a chain of free associations. A better chain, perhaps, is the one in Figure 1 (right). This chain tells the story of Clinton's impeachment and acquittal, the effect on Al Gore's campaign, and finally the elections and the recount. In the following, we identify the properties which make this chain better.

Let us take a closer look at these two chains. Figure 1 (bottom) shows word activation patterns along both chains. Bars correspond to the appearance of a word in the articles depicted above them. For example, the word 'Clinton' appeared throughout the whole right chain, but only at the beginning and the last two articles on the left. It is easy to spot the associative flow of the left chain in Figure 1. Words appear for very short stretches, often only in two neighbouring articles. Some words appear, then disappear for a long period and re-appear. Contrast this with the chain on the right, where the stretches are longer (some words, like Clinton and Lewinsky, appear almost everywhere), and transitions between documents are smoother. This observation motivates our definition of coherence in the next section.

2.2 Formalizing story coherence

Let $\mathcal{D}$ be a set of articles, and $\mathcal{W}$ a set of features (typically words or phrases). Each article is a subset of $\mathcal{W}$. Given a chain $(d_1, ..., d_n)$ of articles from $\mathcal{D}$, we can estimate its coherence from its word activation patterns. One natural

definition of coherence is

$$Coherence(d_1, ..., d_n) = \sum_{i=1}^{n-1} \sum_{w} \mathbb{1}(w \in d_i \cap d_{i+1}).$$

Every time a word appears in two consecutive articles, we score a point. This objective has several attractive properties; it encourages positioning similar documents next to each other and rewards long stretches of words. It is also very simple to compute. However, this objective suffers from serious drawbacks:

Weak links: They say that a chain is only as strong as its weakest link; this applies to our chains as well. Summing over the transitions can lead to 'broken' chains (having weak links), since a chain with many strong links and few weak ones may still score very high. For example, a chain in which all articles but the last one are about the Lewinsky scandal will receive a good score, while not connecting the endpoints in any way.

A more reasonable objective would consider the *minimal* transition score instead of the sum.

$$Coherence(d_1, ..., d_n) = \min_{i=1...n-1} \sum_{w} \mathbb{1}(w \in d_i \cap d_{i+1})$$

However, other drawbacks still exist.

Missing words: Due to our noisy features, some words do not appear in an article, although they should have. For example, if a document contains 'lawyer' and 'court' but not 'prosecution', chances are 'prosecution' is still a highly-relevant word. Considering only words from the article can be misleading in such cases.

Moreover, even if our features were not noisy, an indicator function is not informative enough for our needs.

Importance: Some words are more important than others, both on a corpus level and on a document level. For example, in the shortest-path chain, the first two articles shared several words, among them 'judge' and 'page'. Clearly, 'judge' is more significant, and should affect the objective function more.

Combining **Importance** and **Missing words**, it becomes clear that we need more than a simple word-indicator. Rather, we need to take into consideration the *influence* of d_i on d_{i+1} through the word w. We defer the formal definition of influence to Section 2.3; intuitively, $Influence(d_i, d_j \mid w)$ is high if (1) the two documents are highly connected, and (2) w is important for the connectivity. Note that w does not have to appear in either of the documents. See Figure 2 for an example: the source document d_0 is

d_0 :*Judge Lance Ito lifted his ban on live television coverage of the O.J. Simpson trial*

We calculated word-influence from d_0 to two other documents, using methods explained in Section 2.3. The blue bars (in the back) represent word influence for document

d_1 :*O.J. Simpson's defense lawyers told the judge they would not object to the introduction of DNA evidence*

and the red bars (front) represent word influence for

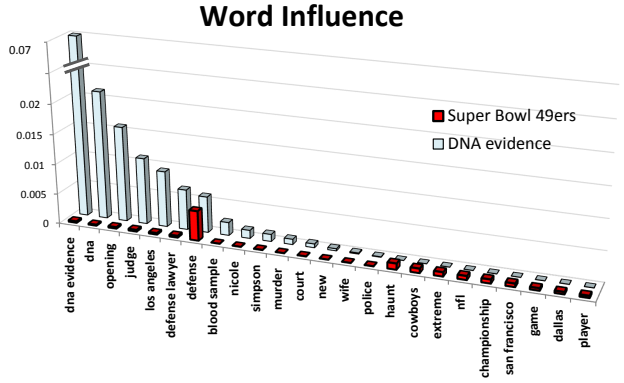

Figure 2: Word influence from an article about the OJ Simpson trial to two other documents – one about football and another about DNA evidence.

d_2 :*Winning three consecutive Super Bowls would be a historic accomplishment for San Francisco 49ers*

First, note that the blue bars are generally higher. This means that d_1 is more relevant to the source article d_0. The influential words for d_1 are mostly court-related, while d_2's are sport-related (interestingly, the word 'Defense' is strong in both documents, for completely different reasons). Note that many of the influential words do not appear in either of the three articles, thereby solving the **Missing words** problem. With the new *Influence* notion, our objective can be re-defined as

$$Coherence(d_1, ..., d_n) = \min_{i=1...n-1} \sum_{w} Influence(d_i, d_{i+1} \mid w)$$

This new objective, while better, still suffers from the problem of **Jitteriness**.

Jitteriness: the objective does not prevent jittery activation patterns, i.e., topics that appear and disappear throughout the chain.

One way to cope with jitteriness is to only consider the longest continuous stretch of each word. This way, going back-and-forth between two topics provides no utility after the first topic switch. Remember, this stretch is not determined by the actual appearance of the word along the chain; words may have a high influence in some transition even if they are missing from one (or both) of the articles. Rather, we define an activation pattern arbitrarily for each word, and compute our objective based on it. The coherence is then defined as the score under the best activation pattern:

$$Coherence(d_1, ..., d_n) = \max_{activations} \min_{i=1...n-1}$$

$$\sum_{w} Influence(d_i, d_{i+1} \mid w) \mathbb{1}(w \text{ active in } d_i, d_{i+1}) \quad (*)$$

Since influence is non-negative, the best solution is to activate all words everywhere. In order to emulate the behaviour of the activation patterns in Figure 1, we constrain the possible activation patterns we consider: we limit the total number of active words and the number of words that are active per transition. In order to avoid multiple stretches, we allow each word to be activated at most once.

Instead of using binary activations (words are either active or inactive), we propose a softer notion of continuous activations. A word's activation is in the range $[0, 1]$, signifying the degree to which it is active. This leads, quite naturally, to a formalization of the problem as a linear program.

2.2.1 *Linear Program Formulation*

The objective function (*) we defined in the previous section can be readily formalized as a linear program (LP). The LP is specified in Figure 3 and illustrated in Figure 4.

max *minedge*

Smoothness

//word w initialized at most once

$$\forall_w \sum_i \textit{word-init}_{w,i} \leq 1 \quad (1)$$

//if w is active in the ith transition,
//either it was active before or just initialized

$$\forall_{w,i} \ \textit{word-active}_{w,i} \leq \textit{word-active}_{w,i-1} + \textit{word-init}_{w,i} \quad (2)$$

//no words are active before the chain begins

$$\forall_w \quad \textit{word-active}_{w,0} = 0 \quad (3)$$

Activation Restrictions

//no more than kTotal words activated

$$\sum_{w,i} \textit{word-init}_{w,i} \leq kTotal \quad (4)$$

//no more than kTrans words active per transition

$$\forall_i \sum_w \textit{word-active}_{w,i} \leq kTrans \quad (5)$$

Objective

//minedge holds the minimum score over edges

$$\forall_i \quad \textit{minedge} \leq \sum_w \textit{word-active}_{w,i} \cdot \textit{influence}(d_i, d_{i+1} \mid w) \quad (6)$$

$$\forall_{w,i} \quad \textit{word-active}_{w,i}, \textit{word-init}_{w,i} \in [0, 1] \quad (7)$$

Figure 3: Scoring a chain.

We are given a chain of n chronologically-ordered documents, $d_1, ..., d_n$. First, we define variables describing word activation levels. We define a variable $\textit{word-active}_{w,i}$ for each document $i = \{1, ..., n-1\}$ and word w. Variable $\textit{word-active}_{w,i}$ measures the activation level of w during the transition from d_i to d_{i+1}. In Figure 4, those variables are represented by the height of the bars. When a word's activation level increases between two consecutive transactions ($d_{i-1} - d_i - d_{i+1}$), we say it was *initialized* in d_i. We define another variable $\textit{word-init}_{w,i}$ indicating the initialization level of w in d_i. In the 0-1 case of Figure 1, $\textit{word-init}_{w,i} = 1$ means that w is first activated in d_i. In the continuous case of Figure 4, $\textit{word-init}_{w,i}$ corresponds to the increase of height between two consecutive transitions.

The LP has three main parts. In **Smoothness**, we require that the activation patterns are smooth: First, constraint (1) requires that each word is activated at most once. Constraint (2) links the initialization and activation variables together. It ensures that an active word w implies that either w was active in the previous transition, or it just got activated. We also set $\textit{word-active}_{w,0} = 0$ (3). Intuitively, it means that no words were active before the beginning of the chain.

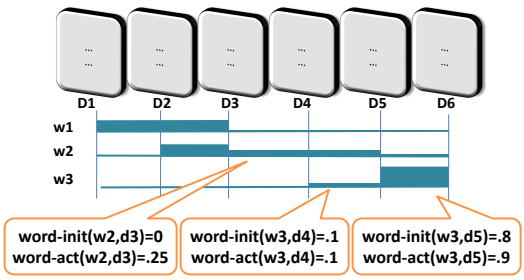

Figure 4: An illustration of the results of the linear program, showing initialization and activation levels along a chain for three words. Activation level is the height of the bars. Initialization level is the difference in activation levels between two consecutive transactions, if the activation level has increased.

In **Activation Restrictions**, we limit the total number of active words (4) and the number of words that can be active during a single transition (5). We use parameters $kTotal$ and $kTrans$ to control the number of active words. The interplay between those two parameters controls the length of activation segments. For example, if $kTotal \sim kTrans \cdot n$, the LP might pick different words for every transition, and segments will be short.

Finally, we get to the **Objective Function**. For every edge i, we calculate its influence. Based on Equation (*), edge influence is the weighted influence of the active words:

$$\sum_w \textit{word-active}_{w,i} \cdot \textit{influence}(d_i, d_{i+1} \mid w)$$

Our goal is to maximize the influence of the weakest link: to do this, we define a variable *minedge*, which takes the minimum influence across all edges (6). Our objective is to maximize this variable.

As a sanity check, we tried the LP on real chains. Figure 5 (left) shows the best activation pattern found for a chain connecting 9/11 and Daniel Pearl's murder (top five words). This pattern demonstrates some of the desired properties from Section 2: the word 'Terror' is present throughout the whole chain, and there is a noticeable change of focus from Bin Laden to Pakistan and the kidnapped journalist. Figure 5 (right) shows activation × influence (rescaled). Notice that words with the same activation levels can have different levels of influence, and thus different effect on the score.

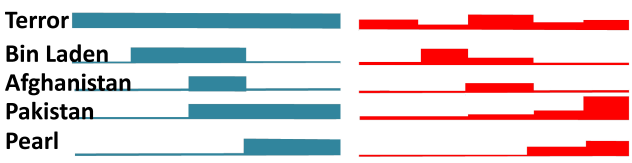

Figure 5: Activation patterns found by our algorithm for a chain connecting 9/11 to Daniel Pearl's murder. Left: activation levels. Right: activation levels weighted by the influence (rescaled). For illustrative purposes, we show the result of the integer program (IP).

2.3 Measuring influence without links

The LP from the previous section required evaluation of *influence*$(d_i, d_j \mid w)$ – the influence of d_i on d_j w.r.t. word w (refer again to Figure 2 for intuition). Several methods for measuring influence have been proposed. The vast majority of them focus on directed weighted graphs (e.g., the web, social networks, citations), where influence is assumed to propagate through the edges. Methods such as authority computation [8], random graph simulations [7] and random walks [3] all take advantage of the edge structure.

However, in our setting no edges are present. Adding artificial edges (formally known as 'link prediction') is a complicated and challenging task. In this section, we explore a different notion of influence; despite the fact that this notion is based on random walks, it requires no edges.

First, we construct a bipartite directed graph, $G = (V, E)$. The vertices $V = V_{\mathcal{D}} \cup V_{\mathcal{W}}$ correspond to documents and words. For every word w in document d, we add both edges (w, d) and (d, w). Refer to Figure 6 for a simple graph: there are four (square) documents, and four (circle) words. The leftmost article, about Clinton admitting Lewinsky liaison, is connected to the words 'Clinton' and 'Judge'.

Edge weights represent the strength of the correlation between a document and a word. The tool we used for word extraction [1] assigns importance to each word; we use these weights for document-to-word edges. Alternatively, we can use TF-IDF weights. Since we interpret weights as random walk probabilities, we normalize them over all words in the document. For example, the rightmost article is mostly (.7) about Al Gore, and somewhat about 'Judge' (.2) and 'Clinton' (.1). The word-to-document weights are computed using the same numbers, but normalized over the documents. The word 'Gore' can only be reached by a single document, so the edge weight is $\frac{.7}{.7} = 1$. We now use this weighted graph to define influence between documents.

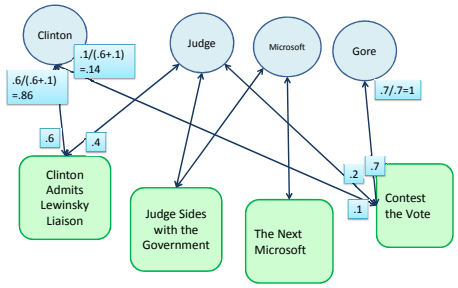

Figure 6: A bipartite graph used to calculate influence.

As mentioned before, *Influence*$(d_i, d_j \mid w)$ should be high if the two documents are highly connected, and w plays an important role in this connection. Intuitively, if the two documents are connected, a short random walk starting from d_i should reach d_j frequently. We first compute the stationary distribution for random walks starting from d_i. We control the expected length with a random restart probability, ϵ. The stationary distribution is the fraction of the time the walker spends on each node:

$$\Pi_i(v) = \epsilon \cdot \mathbb{1}(v = d_i) + (1 - \epsilon)\Sigma_{(u,v)\in E}\Pi_i(u)P(v \mid u)$$

where $P(v \mid u)$ is the probability of reaching v from u.

We now need to factor in the effect of w on these walks. We turn w into a sink node: let $P^w(v \mid u)$ be the same probability distribution as $P(v \mid u)$, except there is no way out of node w. Let $\Pi_i^w(v)$ be the stationary distribution for this new graph. If w was influencial, the stationary distribution of d_j would decrease a lot: in Figure 6, without the word 'Judge' article 1 is no longer reachable from article 2.

The influence on d_j w.r.t. w it defined as the difference between these two distributions, $\Pi_i(d_j) - \Pi_i^w(d_j)$. Figure 2 shows an example of word-influence results calculated by this method. Refer to Section 2.2 for a detailed explanation.

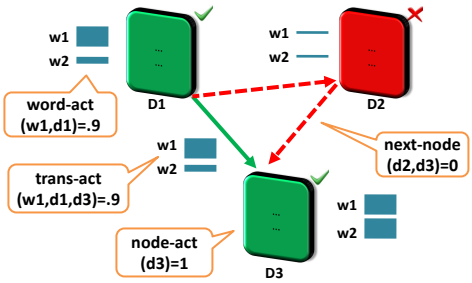

Figure 7: An illustration of the results of the second linear program.

3. FINDING A GOOD CHAIN

In the previous sections we discussed a method to score a fixed chain. However, we are still left with the problem of *finding* a chain. One natural way is to use local search. In local search, we start from a candidate chain and iteratively move to a neighbour chain, chosen to maximize our scoring function. Local search is easy to understand and to implement. However, it suffers from some known drawbacks, in particular a tendency get stuck in a local optimum. In this section we present a different approach. Instead of evaluating many chains along the local-search path, we jointly optimize over words *and* chains.

Similarly to Section 2, we formulate this problem as an LP. The main difference is that neither the transitions nor the articles are known in advance; therefore, we have to consider all articles and edges as candidates for the chain.

Refer to Figure 7 for an illustration of the LP. The figure depicts three articles d_1, d_2, d_3. Articles which are a part of the chain are indicated by a checkmark (in this example, d_1 and d_3). In the LP, this is denoted by variables $\textit{node-active}_i$ (i.e., $\textit{node-active}_1 = 1$, $\textit{node-active}_2 = 0$).

Figure 7 also shows all three possible edges between the articles (since edges are in chronological order, we ignore back-edges). In the figure, the edge from d_1 to d_3 is the only active one, marked by a solid line. (In fact, this is the only solution if d_2 is inactive but d_1 and d_3 are.) Variables $\textit{next-node}_{i,j}$ indicate whether there is a transition from d_i to d_j (i.e., $\textit{next-node}_{1,3} = 1$).

Words have *activation levels* associated with each document. Activation levels are depicted as two bars adjacent to each article, corresponding to words w_1, w_2. For example, w_1 is high in d_1 and d_3. Since d_2 is inactive, both words are inactive in it. Variables $\textit{word-active}_{w,i}$ indicate the activation level of word w in d_i. Note that i previously referred to the ith transition in the chain; since we no longer know the chain in advance, we cannot do this here. Instead, the activation level per transition (bars adjacent to edges) is denoted by variables $\textit{transition-active}_{w,i,j}$. The activation of w_2 along the edge is low, since it was low in d_1.

Like before, the score of an active edge is the sum of acti-

vations along transitions, weighted by influence:

$$\sum_w \textit{transition-active}_{w,i,j} \cdot \textit{influence}(d_i, d_j \mid w)$$

The LP has the same three main parts as before, plus an extra module guaranteeing that a valid chain is formed. Let us look at each of these modules in detail.

Chain Restrictions: This new module ensures a proper chain: starts with s, ends with t, has K nodes (ordered chronologically) and $K-1$ edges. In addition, every node (but s,t) has exactly one incoming and one outgoing edge.

//s and t are in the chain

$$\textit{node-active}_1 = 1, \textit{node-active}_n = 1 \quad (8)$$

//there are K nodes, $K-1$ edges,

$$\sum_i \textit{node-active}_i = K, \ \sum_{i,j} \textit{next-node}_{i,j} = K-1 \quad (9)$$

//Intermediate nodes have one in-edge and one out-edge.
//Inactive nodes have no edges.

$$\sum_i \textit{next-node}_{i,j} = \textit{node-active}_j \quad j \neq s$$

$$\sum_j \textit{next-node}_{i,j} = \textit{node-active}_i \quad i \neq t$$

//the chain is ordered chronologically

$$\forall_{i \geq j} \textit{next-node}_{i,j} = 0 \quad (10)$$

//a transition cannot be active if a middle document is

$$\forall_{i<k<j} \textit{next-node}_{i,j} \leq 1 - \textit{node-active}_k \quad (11)$$

Smoothness: The smoothness module is very similar to the one in Figure 3, connecting the activation and initialization levels of words. The only difference is that we only do not know the transitions in advance.

//word w initialized at most once

$$\forall_w \sum_i \textit{word-init}_{w,i} \leq 1 \quad (12)$$

//word w is active in d_j if it was either initialized at d_j
//or active in d_i, which is previous to d_j in the chain

$$\forall_{w,i,j} \ \textit{word-active}_{w,j} \leq \textit{word-init}_{w,j} + \textit{word-active}_{w,i} + 1 - \textit{next-node}_{i,j} \quad (13)$$

Activation Restrictions: As before, we restrict the number of active words per document and per chain. Moreover, if a word is active in a transition, the transition itself has to be active; the word's activation level cannot exceed its activation level in the originating document.

//no more than kTotal activated words

$$\sum_{w,i} \textit{word-init}_{w,i} \leq kTotal \quad (14)$$

//no more than kTrans words active per document

$$\forall_i \sum_w \textit{word-active}_{w,i} \leq kTrans \quad (15)$$

//if w is active in the transition from i to j,
//it has to be active in i

$$\textit{transition-active}_{w,i,j} \leq \textit{word-active}_{w,i}$$

//a word cannot be active in a non-active transition

$$\textit{transition-active}_{w,i,j} \leq \textit{next-node}_{i,j} \quad (16)$$

Minmax Objective: As before, we define a variable *minedge*, which takes the minimum weight of all active edges. Our goal is to maximize this variable.

//minedge is the minimum of all active edge scores

$$\forall_{i,j} \quad \textit{minedge} \leq 1 - \textit{next-node}_{i,j} + \sum_w \textit{transition-active}_{w,i,j} \cdot \textit{influence}(d_i, d_j \mid w) \quad (17)$$

3.1 Rounding

In order to extract the best chain from the LP optimum, we need a *rounding* procedure. Note that we only need to round the $\textit{node-active}_i$ variables (or alternatively, $\textit{next-node}_{i,j}$). We now present a randomized rounding schema with proven guarantees:

First, we solve the LP. Let $\textit{next-node}^*_{i,j}$ be the value of $\textit{next-node}_{i,j}$ in the optimal solution. The LP solution defines a fractional directed flow of one unit from s to t. We start from node s, and iteratively pick the next node of the chain. The next node is picked proportionally to the flow; in other words, if our current node is d_i, the next node will be d_j with probability $\frac{\textit{next-node}^*_{i,j}}{\sum_j \textit{next-node}^*_{i,j}}$. The flow constraints ensure that this process will stop at t. The ordering constraints ensure that it runs in polynomial time. It is equivalent to a decomposition of the flow into a collection of s-t paths, $\{P_i\}$, and picking a path proportional to its weight (flow).

CLAIM 3.1. *The expected length of a path is K.*

Proof: $E(\textit{path length}) = \sum_i \textit{weight}_i \cdot |P_i| = \sum_v \sum_{\{i:P_i \ \textit{going through node } v\}} \textit{weight}_i = K$

THEOREM 3.2. *Let the optimal value of the LP be V. The value of the rounded solution is at least $(1-c)V$ for $c = \sqrt{\frac{2}{V}\ln(n/\delta)}$ with probability at least $1-\delta$. We can replace n by the number of nodes with non-zero activation in the LP solution.*

Proof Sketch: Define $|\mathcal{W}|+1$ Bernoulli random variables for each edge (i,j) with probabilities $1 - \textit{next-node}^*_{i,j}$ and $\textit{transition-active}^*_{w,i,j}$ for each word w (stars denote the LP value). The edge weight is expected sum of the Bernoulli variables. We bound the probability that this weight is less than $(1-c)V$ using Chernoff bound. c was chosen so that the probability is at most $\delta/\binom{n}{2}$. Taking a union bound, we bound the probability that *any* edge is below $(1-c)V$ by δ.

3.2 Scaling up

The joint LP from Section 3 has $O(|\mathcal{D}|^2 \cdot |\mathcal{W}|)$ variables, and therefore is not feasible for a large number of articles. Certainly, it cannot handle the number of news articles put out every day. In this section, we consider practical ways to speed up the computation.

Selecting an initial set of documents As mentioned, our approach may not be practical when the number of documents is large. However, it can be profitably invoked on a carefully and efficiently selected subset of the documents. We consider ways to restrict the number of candidate articles for the s-t chain.

Picking documents similar to s and t works well when s and t are close, but breaks down for complex chains. For example, impeachment is not an important word in s,t of Figure 1, yet we should include these articles in our candidate subset. We propose to use the same bipartite graph from Section 2.3, run random walks starting from s and t,

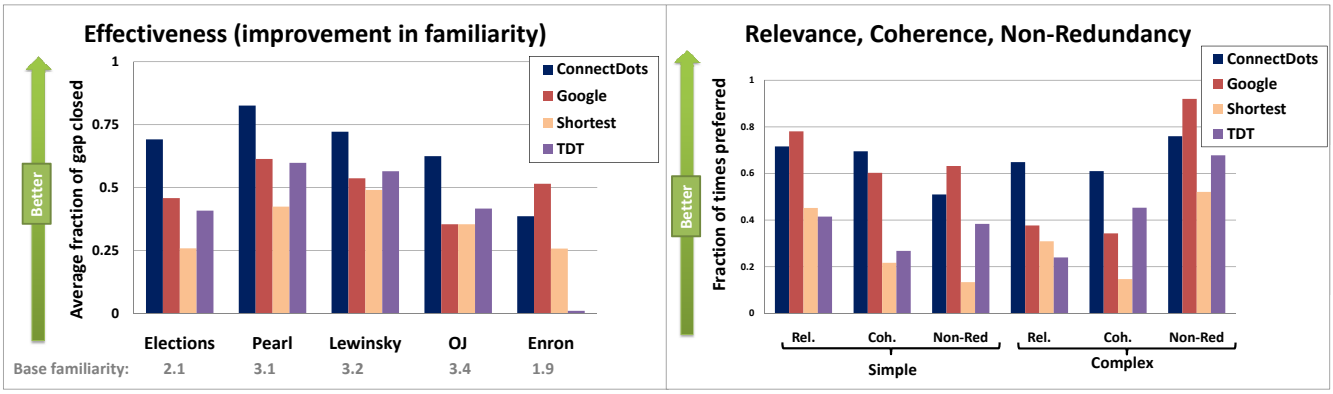

Figure 8: Left: evaulating effectiveness. For each story and each technique, we average over users the fraction of familiarity gap which closed after reading a chain. The number under the story indicates the average familiarity (on a scale of 1 to 5) before reading any chain. Right: Relevance, coherence, and non-redundancy (broken down by simple vs. complex stories). The y axis is the fraction of times each method was preferred, compared to another chain. Users could say both chain are equally good, and therefore the numbers do not sum to 1. Our algorithm outperformed the competitors almost everywhere, especially for complex stories.

and pick the top-ranked articles. Since random walks start from s *and* t, we hope that documents which are frequently reached from both will be ranked high. The same idea may be used to restrict $\mathcal{W}$, the set of words, as well.

We then solve the LP for the restricted set of articles. If the resulting chain is not strong enough, we can iteratively add articles to the set. We add articles from the time period corresponding to the weakest part of the chain, hoping to replace the weak links by stronger ones. This way, we obtain an anytime algorithm which generates a stream of chains, each one chosen from a larger set of candidate articles.

Speeding up influence calculation In Section 2.3, calculating influence required $O(|\mathcal{D}| \cdot |\mathcal{W}|)$ calculations of stationary distributions. We speed up the calculation by using only one set of random walks for all w. For each document d_i we simulate random walks on the original graph. During each walk, we keep track of the word-nodes encountered. When calculating $Influence(d_i, d_j \mid w)$, we only consider the number of times we reached d_j without using w. Thus, we only need $O(|\mathcal{D}|)$ random walks. Note that the random walks are not independent anymore. However, the results are still exact, since we only need the expectations.

4. EVALUATION

Evaluating the performance of information retrieval tasks often focuses on canonical labeled datasets (e.g., TREC competitions) amenable to the standard metrics of precision, recall and variants thereof. The standard methods do not seem to apply here, as they require labeled data, and we are not aware of any labeled dataset suitable for our task. As a result, we evaluated our methods by running them on real data and conducting user studies to capture the utility of our algorithms as they would be used in practice.

We evaluate our algorithm on real news data from the New York Times and Reuters datasets (1995-2003). We preprocessed more than half a million articles. These articles cover a diverse set of topics, including world news and politics, economy, sports and entertainment.

We considered some of the major news stories of recent years: the OJ Simpson trial, the impeachment of Clinton, the Enron scandal, September 11th and the Afghanistan war. For each story, we selected an initial subset of $500 - 10,000$ candidate articles, based on keyword-search. The size of the candidate subset depended on the search results. For example, there were a lot more articles mentioning Clinton than those mentioning Enron.

For each article, we extract named entities and noun phrases using Copernic Summarizer [1]. In addition, the NYT dataset includes important meta-data such as taxonomy and section. We remove infrequent named entities and non-informative noun phrases (e.g., common nouns such as "year").

Our goal was to construct chains representing the stories, and have users evaluate them. For each story, we chose several pairs of articles. We then tried finding stories linking each pair using the following techniques:

- **Connecting-Dots** As described in Section 3, but using the rounding technique we had at the time of the user studies, based on iteratively removing articles.[1] The typical value of K was 6 or 7. $kTotal$ was set to 15, and $kTrans$ was set to 4. We used the speed-up methods of Section 3.2, and allowed ten minutes for the creation of a chain.
- **Shortest-path** We constructed a graph by connecting each document with its nearest neighbours, based on Cosine similarity. If there was no such path, we increased the connectivity of the graph until a path was found. If the path was too long, we picked a subset of K evenly-spaced documents.
- **Google News Timeline** [2] GNT is a web application that organizes news search results on a browsable, graphical timeline. The dataset is different, making comparison hard. Also, the input is a query string. We constructed such a string for each story, based on s and t, and picked K equally-spaced documents between the dates of our original query articles.

[1]During each iteration, we solve the LP from Section 3. We exclude the article with the lowest activation score from the next iterations (setting $node\text{-}active_i = 0$). We stop when exactly K of the $node\text{-}active_i$ variables are set to 1. Since at every iteration we remove one article, the process his is guaranteed to stop after $|\mathcal{D}| - K + 1$ iterations. In practice, it reaches a solution within a few iterations.

Google News Timeline:	**Connect the Dots:**
Osama bin Laden is denounced by his family // Osama Family's Suspicious Site (Web designer from LA buys a bizarre piece of Internet history) // Are you ready to dance on Osama's grave? (How should one react to the death of an enemy?) // Al-Qaeda behind Karachi blast // LIVE FROM AFGHANISTAN: Deadline of Death Delayed for American Journalist // Killed on Job But Spared 'Hero' Label (About Daniel Pearl)	Dispatches From a Day of Terror and Shock // Two Networks Get No Reply To Questions For bin Laden (Coverage of September 11th) // Opponents of the War Are Scarce on Television (Coverage of the war in Iraq and Afghanistan) // 'Afghan Arabs' Said to Lead Taliban's Fight // Pakistan Ended Aid to Taliban Only Hesitantly // Pakistan Officials Arrest a Key Suspect in Pearl Kidnapping (Pearl abducted in Paksitan while investigating links to terror) // The Tragic Story of Daniel Pearl

Figure 9: Example output chains for Connect-Dots and Google News Timeline. Users were given access to the full articles. The GNT chain is a lot less coherent, and includes several insignificant articles.

- **Event threading (TDT)**[13] is a method to discover sub-clusters in a news event and structure them by their dependency, generating a graph. We found a path in this graph from the cluster including s to the cluster including t, and picked a representative document from each cluster along the path. Again, if the chain was too long, we chose K equally-spaced articles.

First, we presented 18 users with a pair of source and target articles. We gauged their **familiarity** with those articles, asking whether they believe they knew a coherent story linking them together (on a scale of 1 to 5). We showed the users pairs of chains connecting the two articles, generated by the above methods in a double-blind fashion. We asked the users to indicate

- **Relevance:** which chain captures the events connecting the two articles better?
- **Coherence:** which chain is more coherent?
- **Redundancy:** which has more redundant articles?

In addition, we measured the **effectiveness** of the chains. We asked users to estimate how their answer to the **familiarity** question changed after reading each chain. **Effectiveness** is the fraction of the familiarity gap closed. For example, if the new familiarity is 5, this fraction is 1 (gap completely closed). If the familiarity did not change, the fraction is 0. This was meant to test whether users feel that the chain helped them gain better understanding of the big picture, which is, after all, our main goal.

Example output chains are shown in Figure 9. Figure 8 shows the results of our user-study. After analyzing the results, we identify two types of stories: *simple* and *complex*. Simple stories tend to focus around the same event, person or institution (e.g., the OJ Simpson trial/ the Enron story). Those stories can usually be summarized by a single query string. In complex stories, however, the source and target article are indirectly connected through one or more events (e.g., Lewinsky-impeachment-elections, September 11th-Afghanistan war-Daniel Pearl).

The left plot shows the **effectiveness** (closing the **familiarity** gap) for each of the methods. Underneath each story we display the average familiarity score before reading any chain (e.g., the Enron story is not well-known).

Our algorithm does better than the competitors on all stories but Enron. The difference is especially pronounced for complex stories. In simple stories, such as Enron, it seems that the simple method of picking K evenly-spaced documents from GNT was sufficient for most people. However, when the story could not be represented as a single query, the effectiveness of GNT decreased.

Surprisingly, GNT did a lot worse on the OJ story than on Enron (note that its score is lower despite the smaller gap). A closer examination revealed that there were a lot more stories about OJ, many of them esoteric at best, so picking random K documents tended to produce poor results (a book of a former juror made it to the best-selling list, etc). Furthermore, more of our users were familiar with the OJ story beforehand, so there was less room for improvement.

As expected, shortest path did badly. Event threading did somewhat better; however, for simple stories, sometimes the clusters were too big. In the Enron story, both s and t belonged to the same cluster, rendering the chain useless. Also, the fact that we pick a representative for each cluster at random might have hurt its performance.

The plot on the right shows the percentage of times each method was credited for relevance, coherence and non-redundancy. Simple stories are grouped on the left, complex – on the right. Users could credit one chain, both or neither. Therefore, the numbers do not sum to 100%. Our algorithm is amongst the best in all measures at a statistically significant level. Most importantly, it achieves the best coherence scores (especially in the complex case). We discuss some of the interesting findings below.

Relevance and Redundancy: As expected, for all methods, relevance is good for simple stories but achieving low redundancy is harder. There is a tradeoff – redundancy is easy to avoid by picking random, possibly irrelevant articles. Relevance is easy to achieve by picking articles similar to s or t, but then redundancy would be high.

Google News Timeline is doing well in terms of relevance for simple stories. However, the chains it generates tend to include somewhat insignificant articles, especially for complex stories. The clusters of Event Threading seem to reduce its redundancy, compared to shortest-path.

Coherence: Together with **effectiveness**, this is perhaps our most important metric. Our algorithm outperforms the other methods, especially in the complex case. This indicates that the notion of coherence devised in this paper matches what the actual users perceive. Interestingly, event threading outperformed GNT for complex stories. This is because the GNT keywords were based on s and t, and did not capture the intermediate events.

5. INTERACTION MODELS

Thus far, we have defined a way to find chains connecting two endpoints. However, the user may not find the resulting chain satisfactory. In information retrieval systems, the solution is often to let the users revise their queries; for a complex information need, users may need to modify their query many times. In this section, we propose to take advantage of the structured nature of the chains, and explore more expressive forms of interaction. We explore two different types of user feedback: *refinement* of a chain, and *tailoring* to user interests.

Refinement: When presenting a chain to a user, some of the links in the chain may not be obvious. Moreover, the user might be especially interested in a specific part of the chain. For example, a user not familiar with the details of the Lewinsky story might want to further expand the first link of Figure 1 (right). We provide the user with a mechanism to indicate areas in the chain which should be further refined; a refinement may consist of adding a new article, or replacing an article which seems out of place.

Since evaluating a single chain is quick, the refinment process is very efficient. Starting from the original chain, we try all possible replacement/insertion actions. We evaluate each chain (see Section 2), and return the best one.

Simpson Defense Drops DNA Challenge
Issue of Racism Erupts in Simpson Trial
Ex-Detective's Tapes Fan Racial Tensions in Los Angeles
Many Black Officers Say Bias Is Rampant in LA Police Force
With Tale of Racism and Error, Lawyers Seek Acquittal
⋆ **In the Joy Of Victory, Defense Team Is in Discord** ⋆
⋆ (Defense lawyers argue about playing the race card) ⋆
The Simpson Verdict

Figure 10: Chain refinement. The starred article was added in order to strengthen the last link.

In Figure 10, the starred article is the result of an insertion request. Adding the article strengthened the end of the chain, while maintaining the global theme.

Incorporate user interests: There can be many coherent ways to connect s and t, especially when they are about similar topics. For example, consider the OJ Simpson trial story. Suppose the user is interested in the racial aspects of the case, but our algorithm finds a chain focusing on the verdict. We provide a mechanism for the user to focus the chains around concepts they find important. In our case, the user may increase the importance of 'racial' or 'black', and perhaps decrease the importance of 'verdict'.

In order to take user's feedback into account, we augment our objective with importance weight π_w for each word w:

$$\sum_w \pi_{\mathbf{w}} \mathit{Influence}(d_i, d_{i+1} \mid w) \mathbb{1}(w \ \mathit{active\ in}\ d_i, d_{i+1})$$

π_w are initialized to 1. When a user likes a word, its importance is increased by a multiplicative factor. When they dislike a word, its weight is decreased by a (perhaps different) factor. These factors were determined empirically, and may change over time (similar to online learning techniques). In order to avoid having to click many words, we use word co-occurrence information to distribute the effect amongst related words. E.g., when a user clicks on 'DNA', the words 'blood' and 'evidence' increase a little too.

Figure 11 shows an actual output of the system. The figure depicts four chains: for each, it shows activation levels for the most important words and a few selected articles. The top-left chain (before any interaction took place) focuses on the verdict. The other chains are derived from it by increasing the weight of 'Black' (top right) or 'DNA' (bottom left). The bottom-right chain is the result of increasing the weight of 'DNA' twice. As can be seen, increasing the weight of a word causes the chain to change its focus. The 'Black' chain focuses on racial issues, and the 'DNA' chains focus more and more on the testimony of the DNA expert.

Another way to interpret user's feedback is as a *constraint*. Under this interpretation, the user imposes constraints on

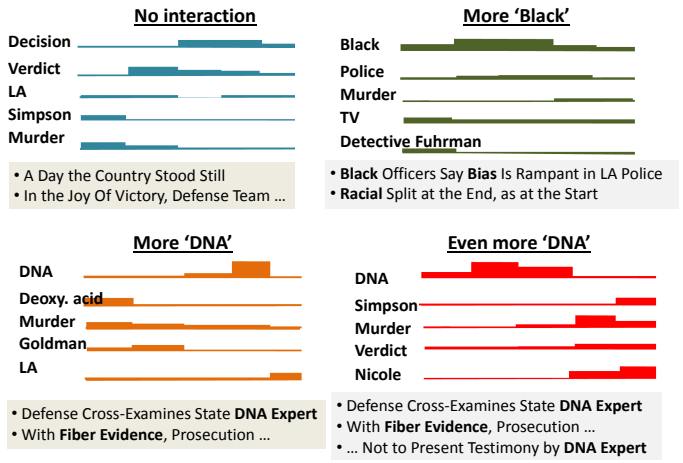

Figure 11: A demonstration of our interactive component. The original chain is at the top left. The rest are derived from it by requesting the words 'black', 'DNA', and 'DNA'×2. For each chain we show activation levels for the most important words, and a few selected articles.

the accumulated influence of specific words in the chain. We do not describe this mechanism in detail; intuitively, if the user indicates that they want more of a word w, the resulting chain will demonstrate higher levels of influence for w, compared to the previous chain. The level of influence measures how much the chain is about w. Note that this is a property of the chain, and does not depend on the activation levels. By letting users indicate desired influece levels, they can change the focus of the chains.

Combining interaction types: The idea of personal word-preferences might also be useful for the **refinement** task. Suppose the user asked to replace an article d_i; if there are many articles similar to d_i, local search is likely to return one of them. We can implement a mechanism similar to our work in [5] to find words which might be attributed for the user's dislike of d_i, and decrease their importance. This way, the replacement article will not be similar.

5.1 Evaluation

We conducted another user study to evaluate how well our algorithm personalizes the chains it selects in response to user feedback. We tested both aspects of feedback:

Refinement: We showed the user a chain, and asked them to perform a refinement operation (asking for insertion/replacement). We then returned two chains, obtained from the original chain by (1) our local search, (2) adding an article chosen randomly from a subset of candidate articles (Section 3.2), obeying chronological order. We asked the user to indicate which chain better fit their request. Users preferred the local-search chains 72% of the time.

User Interests: We showed users two chains – one obtained from the other by increasing the importance of 2-3 words. We then showed them a list of ten words containing the words whose importance we increased and other, randomly chosen words. We asked which words they would pick in order to obtain the second chain from the first. Our goal was to see if users can identify at least some of the words. Users identified at least one word 63.3% of the times.

6. RELATED WORK

To the best of our knowledge, the problem of connecting the dots is novel. There has been extensive work done on related topics, from narrative generation to identifying and tracking news events.

The *narrative generation* community [16, 14] has sought to explore the notion of narratives and the ways to model them. However, their task seems to be fundamentally different. Much of the work involves producing natural-language experiences for a user (e.g., a computer game), and focus on planning-like operators and inferences. In contrast, we do not try to generate any natural-language content, neither do we make up new plots. Our contribution lies in finding a good chain of documents within a given dataset. Nevertheless, some of the work done on evaluating narratives [15] may be useful for our purposes.

For *event tracking*, some efforts have been made to classify news stories into broad categories using pattern matching and machine learning [11]. However, these methods assume the labels are known in advance, and thus are not applicable. *Event detection* [9, 19] deals with discovering new events, but does not attempt to string different events together.

In contrast, email threading [10] tries to discover connections between related email messages. This is closer to the task we have in mind, but much easier, since email usually incorporates a strong structure of referenced messages.

In a work closest to ours, [13, 12] studied how to discover sub-clusters in a news event and structure them by their dependency, generating a graph structure. However, they do not address the notion of coherence at all; constructing a chain of coherent articles from the output graph is hard, as we have seen in the experimental section. In addition, it seems like the method is best-suited for simple news stories, i.e., stories that can be summarized in one or two keywords ("tsunami", in [12]). It is not clear how well this method does for more complex stories.

Our work differs from most previous work in two other important aspects – **expressing information needs** and **structured output and interaction**. Often, users know precisely what they want, but it is not easy for them to distill this down into a few keywords. Our system's input method (related articles) might facilitate this task. Our system's output is interesting, too – instead of the common list of relevant documents, our output is more structured: a chronological chain of articles, and the flow of influences along it. Often, visually exploring a system's results and interacting with it can reveal new and interesting phenomena.

7. CONCLUSIONS AND FUTURE WORK

In this paper we describe the problem of connecting the dots. Our goal is to help people fight information overload by providing a structured, easy way to navigate between topics. We explored different desired properties of a good story, formalized it as a linear program, and provided an efficient algorithm to connect two articles. Finally, we evaluated our algorithm over real news data via a user study, and demonstrate its effectiveness compared to other methods, such as Google News Timeline.

Our system is unique in terms of input and output, and incorporating feedback into it allows users to fully exploit its capabilities. In the future, we plan to explore richer forms of input and output, allowing for more complex tasks, e.g., creating a roadmap – a set of intersecting chains that covers a topic from several aspects.

In addition, we plan to explore the behaviour of our system under different query characteristics. For example, in our evaluation we considered news stories which were associated with popular events. It would be interesting to test the approach would work for news articles which do not have as much coverage within the news corpus.

We believe that the system proposed in this paper may be a promising step in the battle against information overload. The ability to connect two pieces of information and form a logical, coherent story has applications in many areas. Perhaps most importantly, significant scientific discoveries can come from forming connections between different fields. We plan to extend our methods to scientific papers; we believe that tools to automatically connect the dots can be a great vehicle to enable new discoveries.

Acknowledgements: The authors would like to thank the reviewers for their comments. This work was partially supported by ONR YIP N00014-08-1-0752, ARO MURI W911NF0810242, and NSF Career IIS-0644225. Dafna Shahaf was supported in part by Microsoft Research Graduate Fellowship.

8. REFERENCES

[1] Copernic, http://www.copernic.com.
[2] Google news timeline, http://newstimeline.googlelabs.com/.
[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems*, 1998.
[4] R. Choudhary, S. Mehta, A. Bagchi, and R. Balakrishnan. Towards characterization of actor evolution and interactions in news corpora. In *Advances in Information Retrieval*.
[5] K. El-Arini, G. Veda, D. Shahaf, and C. Guestrin. Turning down the noise in the blogosphere. In *KDD '09*, 2009.
[6] E. Gabrilovich, S. Dumais, and E. Horvitz. Newsjunkie: providing personalized newsfeeds via analysis of information novelty. In *WWW '04*, 2004.
[7] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD '03*.
[8] J. Kleinberg. Authoritative sources in a hyperlinked environment, 1999.
[9] J. Kleinberg. Bursty and hierarchical structure in streams, 2002.
[10] D. D. Lewis and K. A. Knowles. Threading electronic mail: A preliminary study. *Information Processing and Management*, 33, 1997.
[11] B. Masand, G. Linoff, and D. Waltz. Classifying news stories using memory based reasoning. In *SIGIR '92*, 1992.
[12] Q. Mei and C. Zhai. Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In *KDD '05*, 2005.
[13] R. Nallapati, A. Feng, F. Peng, and J. Allan. Event threading within news topics. In *CIKM '04*, 2004.
[14] J. Niehaus and R. M. Young. A computational model of inferencing in narrative. In *AAAI Spring Symposium '09*, 2009.
[15] J. P. Rowe, S. W. McQuiggan, J. L. Robison, D. R. Marcey, and J. C. Lester. Storyeval: An empirical evaluation framework for narrative generation. In *AAAI Spring Symposium '09*, 2009.
[16] S. R. Turner. The creative process: A computer model of storytelling and creativity, 1994.
[17] C. Yang, X. Shi, and C. Wei. Tracing the event evolution of terror attacks from on-line news. In *Intelligence and Security Informatics*.
[18] Y. Yang, T. Ault, T. Pierce, and C. Lattimer. Improving text categorization methods for event tracking. In *SIGIR '00*, 2000.
[19] Y. Yang, J. Carbonell, R. Brown, T. Pierce, B. Archibald, and X. Liu. Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems*, 14(4), 1999.