# Connecting Two (or Less) Dots: Discovering Structure in News Articles

DAFNA SHAHAF and CARLOS GUESTRIN
Carnegie Mellon University

Finding information is becoming a major part of our daily life. Entire sectors, from Web users to scientists and intelligence analysts, are increasingly 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 article, 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 health care debate (2009).

We formalize the characteristics of a good chain and provide a fast search-driven algorithm to connect two fixed endpoints. We incorporate user feedback into our framework, allowing the stories to be refined and personalized. We also provide a method to handle partially-specified endpoints, for users who do not know both ends of a story. Finally, we evaluate our algorithm over real news data. Our user studies demonstrate that the objective we propose captures the users' intuitive notion of coherence, and that our algorithm effectively helps users understand the news.

Categories and Subject Descriptors: I.2.6 [**Artificial Intelligence**]: Learning; G.3 [**Probability and Statistics**]:

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Coherence

**ACM Reference Format:**
Shahaf, D. and Guestrin, C. 2012. Connecting two (or less) dots: Discovering structure in news articles. ACM Trans. Knowl. Discov. Data 5, 4, Article 24 (February 2012), 31 pages.
DOI = 10.1145/2086737.2086744 http://doi.acm.org/10.1145/2086737.2086744

## 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



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 a Microsoft Research Graduate Fellowship.
Author's address: D. Shahaf; email: dshahaf@cs.cmu.edu.

DOI 10.1145/2086737.2086744 http://doi.acm.org/10.1145/2086737.2086744

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 article, we investigate methods for automatically connecting the dots—providing a structured, easy way to uncover hidden connections between two pieces of information.

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 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 of news articles 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 health care, 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.

We believe that the ability to connect dots and form a logical, coherent chain lies at the basis of understanding a topic. We view chains as good building blocks for more complex structures, such as graphs. Existing graph structures (e.g., Nallapati et al. [2004]) often use single edges as their main building blocks, and therefore they are limited to only encoding local interactions. Using chains, we can construct graphs where entire paths are coherent, thus achieving a more global behavior.

There are multiple ways that connecting dots can be utilized. It can be used as an information discovery tool, indicating links between two topics of interest; as an educational tool, pointing out what is appropriate to learn next; or as a context-providing tool, explaining what led to a certain event. In the following, we outline two use-case scenarios.

*Use-Case 1 Browse.* Many news sites present a "Related Articles" feature, indicating articles that the user may find interesting. We propose to augment this feature by

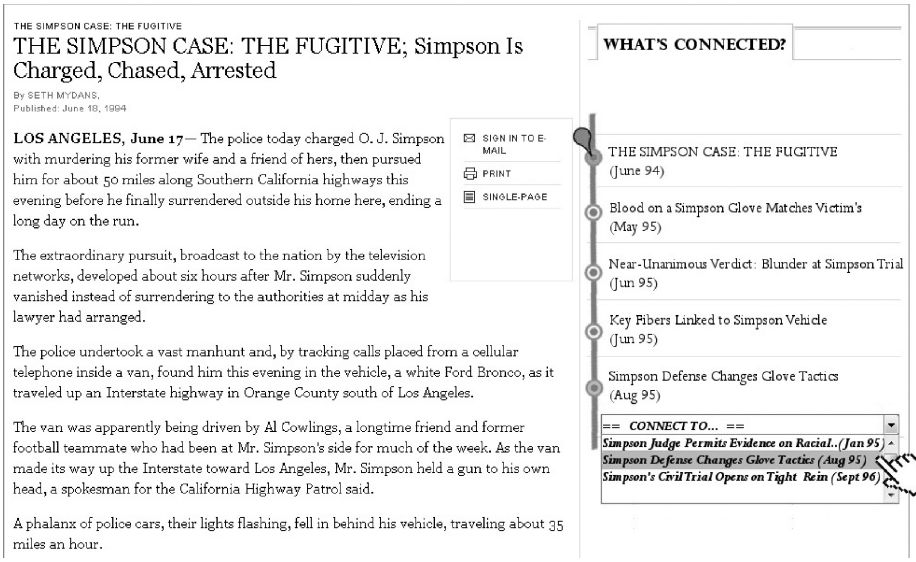



Fig. 1. Use-Case 1 (Illustration): Adding a Connect-the-Dots sidebar to a news site allows the user to create chains between the current article (top right) and other articles. Candidate articles are chosen from a drop-down menu (bottom right). After an article is chosen, the system computes the most coherent chain between the two articles and displays it to the user (right).

a Connect-the-Dots mechanism, allowing the user to connect the current article to related articles, and see it in the context of coherent chains.

Consider Figure 1 for an example. The user is browsing an article about the OJ Simpson trial on the New York Times Web site. The sidebar on the right features the Connect-the-Dots mechanism. At first, the user sees only the current article (marked). He can then choose the other end of the chain from a drop-down menu. Once an article is picked from the menu, the algorithm quickly computes a coherent chain connecting both articles and displays it to the user.

In Figure 1 the user chose an article about the bloody gloves found at the crime scene. This choice resulted in a chain focusing on evidence in the Simpson trial, including fiber and DNA evidence. The user can then browse the articles, or choose to construct a different chain.

We believe that this feature will allow news readers to grasp important aspects of the news quickly and easily.

*Use Case 2 Search.* Information search is one of the most common tasks users are performing on the Internet. The output of search engines today tends to consist of a list of relevant documents; a few group similar results together into clusters. However, building on top of chains, we can conceive richer forms of output.

For example, consider a newssite search engine whose output is a set of (possibly intersecting) coherent chains, covering diverse and important aspects of the query. Visually exploring the chains can reveal new and interesting connections, which are harder to see in a textual list.

To the best of our knowledge, the problem of connecting the dots is novel. Previous research (e.g., Choudhary et al. [2008]; Gabrilovich et al. [2004]; Nallapati et al. [2004]; Yang et al. [1999, 2000, 2006]) focused on organizing news articles into hierarchies or graphs, but did not address the notion of output coherence.

Our main contributions are:[1]

— formalizing characteristics of a good chain of articles and the notion of coherence;
— formalizing influence with no link structure;
— providing an efficient algorithm for connecting two fixed endpoints while maximizing chain coherence;
— providing an efficient interactive algorithm for the case of partially-specified endpoints, based on the notion of coverage;
— incorporating feedback and interaction mechanisms into our system, tailoring stories to user preferences; and
— evaluating our algorithm over real news data and demonstrating its utility to newsreaders via a user study.

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 Chain of Articles Good?

Our goal is to find a good path between two articles, $s$ and $t$. We refer to this path as a story between $s$ 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 2 (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 2.

The problem seems to lie with the locality of shortest-path. Every two consecutive articles are related, but there is no global, coherent theme to the chain as a whole. Rather, shortest-path may exhibit stream-of-consciousness behavior, linking $s$ and $t$ by a chain of free associations. A better chain is in Figure 2 (right). This chain tells the story of Clinton's impeachment and acquittal, the effect on Al Gore's campaign, and finally the elections and recount. In the following, we identify the properties that make this chain better.

Let us take a closer look at these two chains. Figure 2 (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 2. Words appear for very

[1]This article is an expanded version of [Shahaf and Guestrin 2010], including new algorithms, proofs, and a detailed discussion of proper parameter choice. This article also poses a new variant of the problem (partially-specified endpoints) and proposes an interactive algorithm to solve it.

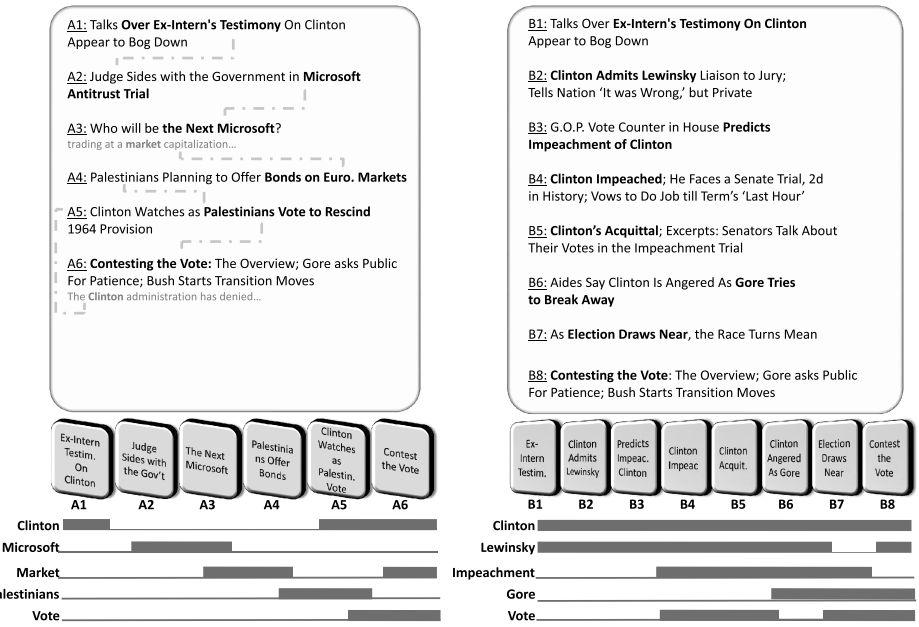



Fig. 2. Two examples of stories connecting the same endpoints. Left: chain created by shortest-path (dashed lines indicate similarities between consecutive articles). Right: a more coherent chain. Activation patterns for each chain are shown at the bottom; the bars indicate appearance of words in the article above them.

short stretches, often only in two neighboring articles. Some words appear, then disappear for a long period and reappear. 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 Chain 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.

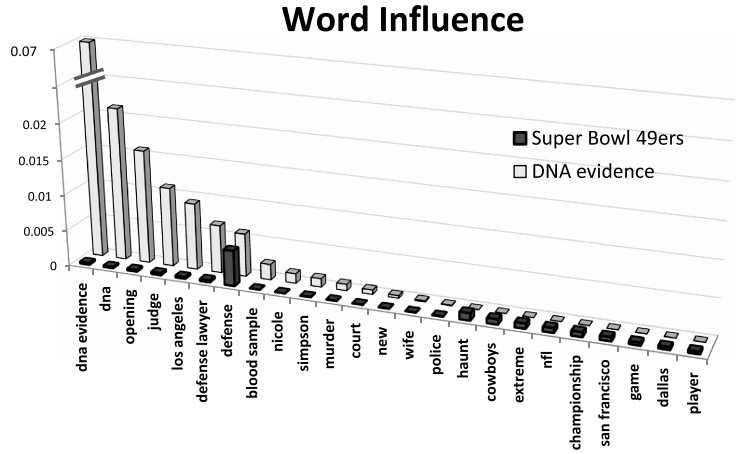



Fig. 3. Word influence from an article about the OJ Simpson trial to two other documents – one about football and another about DNA evidence.

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 3 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

**$d_2$** : 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 any of the three articles, thereby solving the Missing words problem. With the new Influence notion, our objective can be redefined 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: 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 nonnegative, the best solution is to activate all words everywhere. In order to emulate the behavior of the activation patterns in Figure 2, 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 5 and illustrated in Figure 4.

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 $word\text{-}active_{w,i}$ for each document $i = \{1, ..., n-1\}$ and word $w$. Variable $word\text{-}active_{w,i}$ 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 $word\text{-}init_{w,i}$ indicating the initialization level of $w$ in $d_i$. In the 0-1 case of Figure 2, $word\text{-}init_{w,i} = 1$ means that $w$ is first activated in $d_i$. In the continuous case of Figure 4, $word\text{-}init_{w,i}$ corresponds to the increase of height between

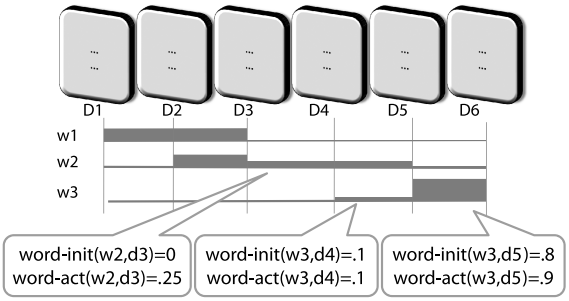



Fig. 4. An illustration of the results of the linear program, showing initialization and activation levels along a chain for three words. Activation level is the height of the bars. Initialization level is the difference in activation levels between two consecutive transactions, if the activation level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

$$\max \ minedge$$

**Smoothness**

*//word $w$ initialized at most once*

$$\forall_w \ \sum_i word\text{-}init_{w,i} \leq 1 \tag{1}$$

*//if $w$ is active in the ith transition,*

*//either it was active before or just initialized*

$$\forall_{w,i} \ word\text{-}active_{w,i} \leq word\text{-}active_{w,i-1} + word\text{-}init_{w,i} \tag{2}$$

*//no words are active before the chain begins*

$$\forall_w \quad word\text{-}active_{w,0} = 0 \tag{3}$$

**Activation Restrictions**

*//no more than kTotal words activated*

$$\sum_{w,i} word\text{-}init_{w,i} \leq kTotal \tag{4}$$

*//no more than kTrans words active per transition*

$$\forall_i \quad \sum_w word\text{-}active_{w,i} \leq kTrans \tag{5}$$

**Objective**

*//minedge holds the minimum score over edges*

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

$$\forall_{w,i} \quad word\text{-}active_{w,i}, word\text{-}init_{w,i} \in [0,1] \tag{7}$$

Fig. 5. Scoring a chain.

two consecutive transitions. Note that in the continuous case, a word can be initialized several times.

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,


Fig. 6. Activation patterns found by our algorithm for a chain connecting 9/11 to Daniel Pearl's murder. (a) Activation levels. (b) Activation levels weighted by the influence (rescaled). For illustrative purposes, we show the result of the integer program (IP) we get replacing constraint (7) of the LP by its binary equivalent.

or it just got activated. We also set $word\text{-}active_{w,0} = 0$, (3). Intuitively, it means that no words were active before the beginning of the chain.

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 the activation segments. For example, if $kTotal \sim kTrans \cdot (K-1)$, the LP might pick different words for every transition, and segments will be short. See Section 3.2 for a detailed discussion.

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.

$$\sum_w word\text{-}active_{w,i} \cdot influence(d_i, d_{i+1} \mid w).$$

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 6 (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 6 (right) shows activation $\times$ influence (rescaled). Notice that words with the same activation levels can have different levels of influence, and thus different effects on the score.

### 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 $d_j$ with respect to word $w$ (refer again to Figure 3 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 [Kleinberg 1999], random graph simulations [Kempe et al. 2003], and random walks [Brin and Page 1998], 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 7 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."

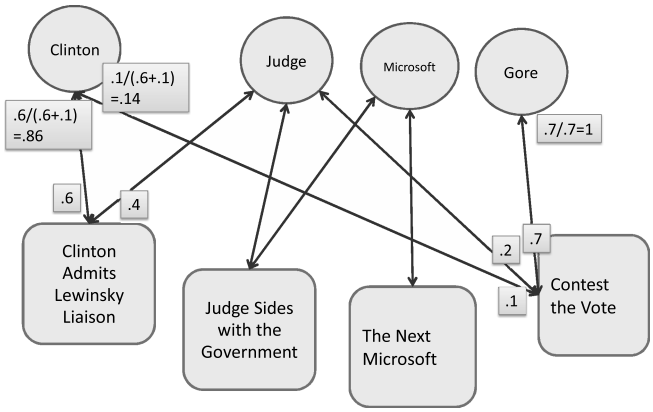



Fig. 7. A bipartite graph used to calculate influence.

Edge weights represent the strength of the connection between a document and a word. Copernic[2], the tool we used for word extraction, assigns importance to each word; we found that these weights worked well as document-to-word edge weights. Unfortunately, the Copernic algorithm is proprietary, so its precise details were not given. Alternatively, one can use TF-IDF weights. Note that unlike Copernic, TF-IDF requires us to carefully choose the corpus.

Since we later interpret weights as random walk probabilities, we normalize the weights 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.

As mentioned before, $\mathit{Influence}(d_i, d_j \mid w)$ should be high if the two documents are strongly 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, $\epsilon$. 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 $u$.

Intuitively, if $d_i$ and $v$ are very related, $\Pi_i(v)$ is high, as many walks reach $v$. We now need to factor in the effect of word $w$ on these walks. In particular, we are interested in knowing how many of the walks went through word $w$ before reaching $v$. To do this, 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 $w$.

$$P^w(v \mid u) = \begin{cases} 0 & \text{if } u = w \text{ and } v \neq w \\ 1 & \text{if } u = v = w \\ P(v \mid u) & \text{o/w} \end{cases}.$$

[2]http://www.copernic.com

Table I. Top 10 Influential Words for Different Values of $\epsilon$

| **0.01** | **0.25** | **0.99** |
|---|---|---|
| glove | glove | glove |
| apparel | apparel | apparel |
| David Margolick | Richard Rubin | David Margolick |
| Judge Lance Ito | Aris Isotoner | murder |
| Ronald Goldman | Michael Conners | Ronald Lyle Goldman |
| Mark Fuhrman (detective) | video recording | Simpson murder |
| black | David Margolick | Los Angeles |
| jury system | Marcia Clark (deputy dist. atty.) | Nicole Brown Simpson |
| Kenneth Noble | Richard Rubin | Simpson |
| police | Bloomingdales | Johnny Cochran Jr. |

Let $\Pi_i^w(v)$ be the stationary distribution for this new graph.

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

If $w$ was influential, the stationary distribution measured at $d_j$ would decrease a lot. In Figure 7, without the word "Judge," article 1 (leftmost) is no longer reachable from article 2. Therefore, we define the influence on $d_j$ with respect to $w$ as the difference between these two distributions.

$$\mathit{Influence}(d_i, d_j \mid w) = \Pi_i(d_j) - \Pi_i^w(d_j). \tag{8}$$

We calculate both stationary distributions, $\Pi_i(v)$ and $\Pi_i^w(v)$, by the power method, and then take their difference. Note that difference is better than quotient, since we want high influence to imply that the two documents are strongly connected. Each iteration of the power method is quadratic in the number of nodes in the graph.

Example word-influence results that were calculated according to Equation (8) appear in Figure 3. Refer to Section 2.2 for a detailed explanation of the figure.

*2.3.1 The Effect of $\epsilon$.* The random walk results depend a lot on the choice of $\epsilon$. $\epsilon$ controls the expected length of a walk: in expectation, we experience a random restart every $\frac{1}{\epsilon}$ steps.

Prescribing the optimal random-walk length is hard. Each walk should be long enough to explore the natural cluster of its starting point $d_i$, but not long enough to forget where it came from. In particular, lists of influential words created by very long walks tend to include words that are unrelated to $d_i$. On the other hand, very short walks tend to only produce words from the immediate neighborhood of $d_i$, losing higher-order word cooccurrences in the process.

Table I further demonstrates the effect of $\epsilon$. We have computed the influence between two articles about OJ Simpson's trial. The articles discussed the bloody gloves recovered at the scene of the murder. Table I shows the top ten influential words for different values of $\epsilon$ for the sake of demonstration, we show the value used in our experiments (0.25), and two extreme values (0.01 and 0.99).

First, we note that the top two words ("glove" and "apparel") are the same across the table. This is because these words appear in both $d_i$ and $d_j$. In addition, they are among the most important words in $d_i$ (as reflected by their importance scores). Thus, many of the random walks from $d_i$ to $d_j$ have gone through these words.

When $\epsilon = 0.01$ (long walks), the list contains many words that are ubiquitous throughout the Simpson story, but not necessarily related to the glove episode (e.g. "Judge Ito," "Ronald Goldman," "jury," and "police"). As expected, prevalent aspects of

the story are heavily represented. For example, the words "black" and "detective Mark Fuhrman" are related to the racial tension of the Simpson case.

On the other hand, when $\epsilon = 0.99$ (short walks), the list contains mostly words that appear directly in $d_i$ (but not necessarily in $d_j$). When we examine the influence scores, we first notice that they are all very low. This is to be expected, since the probability of reaching $d_j$ is lower to begin with. However, even after normalizing the influences, we still observe a very large gap between the first two words and the rest of the list. As noted before, these words ("glove" and "apparel") appear in both $d_i$ and $d_j$, and thus participate in a path of length two. Longer paths are much less likely when $\epsilon$ is high, and thus words which lie on those longer paths did not get to play an important role in the walks.

Finally, $\epsilon = 0.25$ seems to achieve a healthy balance: the list of words includes mostly words that are directly related to the glove story, but do not necessarily appear in either $d_i$ or $d_j$. For example, Richard Rubin is a glove designer who testified that leather gloves shrink when exposed to liquid, Aris Isotoner is the company that manufactured the gloves, and Michael Conners is a photographer who took a picture showing OJ Simpson, several years before the trial, wearing a pair of gloves resembling those found at the crime scene.

## 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 neighbor 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. Our weakest-link objective creates a plethora of local optima, thus aggravating the problem.

In Shahaf and Guestrin [2010], we presented a different approach. We have formulated this problem as an LP, and jointly optimized over the choice of words *and* chains. The main difference between this LP and the LP from Section 2.2 is that neither the transitions nor the articles were known in advance; therefore, we had to consider all articles and edges as candidates for the chain. We have also presented a randomized rounding schema with proven guarantees.

The approach in Shahaf and Guestrin [2010] has produced good chains, but generating the chains was a slow process. In addition, this approach only provided an approximate solution. Approximation algorithms are a common technique when dealing with hard problems; however, in our case, the LP objective poses difficulties along this path.

In particular, consider two coherent chains—$(d_1, ..., d_k)$ and $(d_k, ..., d_{2k-1})$. Intuitively, concatenating the chains can result in a much weaker chain; despite the fact that both chains share an article, they may focus on completely different aspects. However, the LP objective implies that

$$Coherence(d_1, .., d_{2k-1}) \geq \frac{1}{2} \min\{Coherence(d_1, ..., d_k), Coherence(d_k, .., d_{2k-1})\},$$

since we can scale down the LP variables of both chains by a factor of two, and the result is a valid solution for the concatenated chain. In other words, concatenating two chains will cause us to lose a factor of two at most. A similar principle holds for concatenating more than two chains. In fact, a greedy algorithm that considers single edges completely out of context, will result in a $\frac{1}{K}$ approximation ratio for a chain of length $K$.

For this reason, we abandon high approximation ratios. Instead, we will now explore practical ways to speed up the process of finding an optimal chain.

### 3.1 The Algorithm

Of all search strategies used in problem solving, one of the most popular methods of exploiting heuristic information to cut down search time is the informed best-first strategy. The general philosophy of this strategy is to use the heuristic information to assess the merit latent in every candidate direction exposed during the search and explore the direction of highest merit first.

Refer to Algorithm 1 for our main algorithm, `findOptimalChain`. The algorithm tries to find the most coherent *s*-*t* chain of length $K$. The outline of the algorithm is based on the general best-first strategy [Dechter and Pearl 1985]. We keep a priority queue of selected chains (line 6). We initialize the queue with the chain consisting only of vertex $s$ (line 7). At each iteration, we expand the chain that features the highest merit (line 9), generating all of its valid extensions (lines 17–18). If the current chain is of length $l$ and ends with vertex $u$, `validExtensions`$(u, l)$ returns the set of neighbors of $u$ in $G$ which can reach $t$ in exactly $K - l - 1$ steps.

We would like to terminate the search as soon as the first *s*-*t* chain of length $K$ is selected for expansion (line 16), but without compromising optimality. In order to do this, it is sufficient to require that the evaluation function used to sort the queue is admissible—always provides optimistic estimates of the final costs. We will use the following lemma.

LEMMA 3.1. *Let* $\pi_k = (d_1, d_2, ..., d_k)$ *a chain, and* $\pi_{k+1} = (d_1, d_2, ..., d_k, d_{k+1})$ *a chain extending* $\pi_k$ *by a single article* $d_{k+1}$. *Then* $Coherence(\pi_k) \geq Coherence(\pi_{k+1})$.

PROOF. Consider the LPs for $\pi_k, \pi_{k+1}$, denoted $LP_k$ and $LP_{k+1}$. We now show that each feasible solution to $LP_{k+1}$ maps to a feasible solution of the same value for $LP_k$.

The variables of $LP_k$ are a subset of the variables of $LP_{k+1}$. We use the simplest mapping: given a solution to $LP_{k+1}$, we assign the same value to all shared variables of $LP_k$. This is a feasible solution: Constraints (2),(3),(5),(6), and the relaxed version of (7) are all satisfied directly by $LP_{k+1}$. Constraints (1) and (4) provide upper bounds on summations of nonnegative numbers. Since the bound holds for $LP_{k+1}$, it must hold when we sum over a subset of the numbers. Therefore, constraints (1) and (4) are satisfied in in $LP_k$ as well.

Finally, *minedge*, the objective variable, is also a shared variable. Thus, there exists a feasible solution to $LP_k$ of the same value. Since this holds for every feasible solution to $LP_{k+1}$, we have shown that $Coherence(LP_k) \geq Coherence(LP_{k+1})$. □

As a direct consequence, any chain extending $\pi$ is always at most as coherent as $\pi$. Therefore, $Coherence(\pi)$ is an optimistic evaluation for each of its extensions, and we can use it without compromising optimality. In lines 11–12 we call `evalLP` to evaluate chains using the LP from Section 2.2.

Using the LP to evaluate chains has many benefits; in particular, solving the LP is very quick, usually taking a fraction of a second. In fact, it is so quick that we can use an Integer Program (IP) solver and solve the problem with binary activations. Still, the lemma hints at another shortcut we may take: when we extend chain $\pi$ with edge $e$, the value of the extended chain cannot exceed

$$\min\{Coherence(\pi), Coherence(e)\}. \tag{9}$$

If $e$ happens to be a weak edge, we do not need to solve the LP for the extended chain in order to know it is also weak.

**Algorithm 1:** findOptimalChain$(G, s, t, K)$

**input** : $G = (V, E)$ graph, $s, t \in V$, $K$ integer.
**output**: A most coherent $s$-$t$ chain of length $K$ using only edges from $E$.

```
1  foreach v ∈ V compute possible distances from s to v and from v to t ;
2  if no s-t chain of length K exists then
3      return ∅ ;
   // Calculate an upper limit for edge coherence
4  foreach (u, v) ∈ E do
5      val_(u,v) = evalEdge(u, v) ;
6  Q = New priority queue ;
7  Insert ⟨(s), 0⟩_exact into Q ;
8  while Q ≠ ∅ do
9      Extract p = ⟨π, val_π⟩ from the top of Q ;
10     if p is marked 'approx' then // Replace with a tighter estimate:
       evaluate with LP of Section 2
11         newval_π = evalLP(π) ;
12         Insert ⟨π, newval_π⟩_exact into Q ;
13     else // p is marked as 'exact'
           // Is it a solution? If not, find valid chain extensions
14         Let u be the last node of π;
15         if u = t then // Found the best s-t chain of length K
16             return π ;
           // Find all neighbours of u which can reach t in K − |π| − 1 steps, and
             insert extended chain into the queue
17         foreach v ∈ validExtensions(u, |π|) do
18             Insert ⟨(π, v), min{val_π, val_(u,v)}⟩_approx into Q ;
```

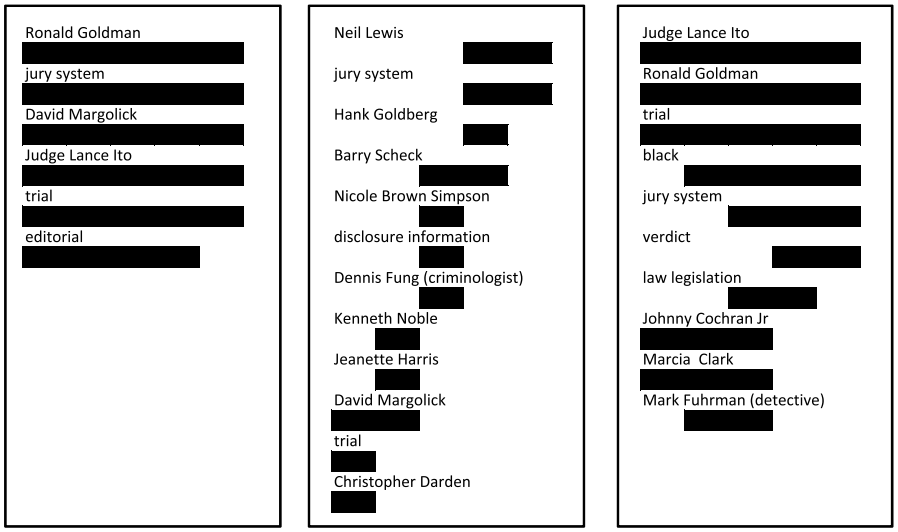
We avoid unnecessary LP computations by caching the coherence of all single edges in advance (line 5). Note that we do not have to solve the LP for single edges. Instead, evalLP$(u, v)$ computes the sum of influences of the top $kTrans$ influential words for edge $(u, v)$. The result is precisely the coherence of the edge, and is very efficient to compute. When we come across a chain for the first time, we evaluate it using Equation (9) (line 18). Only when the chain is chosen for expansion, we replace this approximation with the tighter bound derived from the LP (lines 10–12).

THEOREM 3.2. *Algorithm* findOptimalChain *always terminates with an optimal solution, or returns* $\emptyset$ *if no solution exists.*

PROOF. In line 1, the algorithm computes all possible distances from $s$ and to $t$. The algorithm exits on line 3 and returns $\emptyset$ if and only if $s$ cannot reach $t$ in $K - 1$ steps. By definition, this is exactly the case when no solution exists.

The optimality part of the proof follows from the admissibility of the evaluation function. When the algorithm terminates its search, it has found a chain whose actual coherence is higher than the estimated coherence of any chain that extends any chain in the queue. But since those estimates are admissible (and thus optimistic), we can safely ignore the chains in the queue. □

Each node expanded during the search requires solving a linear program of $O(K \cdot |\mathcal{W}|)$ variables, which can be done in weakly polynomial time. In the worst case, the




(a) $kTotal = kTrans$ (b) $kTotal \gg kTrans$ (c) $kTotal > kTrans$

Fig. 8. Activation levels for the best *s-t* chain for different values of $kTotal$ and $kTrans$.

number of nodes expanded is exponential in $K$. In practice, however, the search is very quick. The following table shows a comparison between `findOptimalChain` and the LP-based solution of Shahaf and Guestrin [2010] in terms of running time and solution quality (percentage of the optimal solution; of course, the solution quality of `findOptimalChain` is always 100%). As can be seen, `findOptimalChain` outputs better chains faster.

| | `findOptimalChain` | | [Shahaf and Guestrin 2010] | |
|---|---|---|---|---|
| | Running Time (s) | Quality | Running Time (s) | Quality |
| Elections | 319 | 100% | 859 | 88% |
| Pearl | 282 | 100% | 1086 | 64% |
| Lewinsky | 92 | 100% | 337 | 79% |
| OJ | 255 | 100% | 714 | 93% |
| Enron | 71 | 100% | 164 | 90% |

### 3.2 The Effect of *kTotal, kTrans*

The LP of Section 2 has introduced two parameters, $kTotal$ and $kTrans$. $kTotal$ restricts the total number of active words, and $kTrans$ restricts the number of active words per transition. The choice of parameters determines valid activation patterns, and thus has a significant effect on the value of chains. To demonstrate the effect of these parameters, we have chosen two articles about OJ Simpson's trial. We have then computed the best chain between the articles for different values of $kTotal$ and $kTrans$.

Without loss of generality, we only consider cases where $kTotal \geq kTrans$. If $kTotal \gg kTrans$ (and in particular, when $kTotal$ approaches $kTrans \cdot (K-1)$), the LP (or IP) can afford to pick $kTrans$ different words for each transition. This results in a behavior similar to the shortest-path chains of Section 2. Figure 8(b) shows

activation levels of the optimal chain. Note the short segments; as in Section 2, they indicate the lack of a global theme.

Setting $kTotal = kTrans$ is equivalent to picking $kTotal$ words to be active throughout the chain. In other words, we are looking for $kTotal$ segments of length $K$. Similar to the way short segments encourage rare words, long segments push towards common words. See Figure 8(a): active words include names of key figures in the trial, and generic words such as "editorial." Note that the last segment is not of full length; this implies that activating the word in the final transition would have no effect on our weakest-link objective.

Medium-length segments (Figure 8(c)) seem to combine the best of both worlds. Transitions are forced to share words between them, but these words do not have to be common across the entire chain. In our experiments, we have found that $\frac{kTotal}{4} \leq kTrans \leq \frac{kTotal}{2}$ often produces good results for $K = 7$. When $kTotal$ is small, the words tend to capture the essence of the story nicely, but the stories themselves tend to be rather simple. As we increase $kTotal$, the stories become more and more complicated. As before, increasing $kTotal$ too much can result in behavior similar to the shortest-path chains.

### 3.3 Scaling Up

Our algorithm is much faster than the LP solution of Shahaf and Guestrin [2010] in practice, but it can take time exponential in the number of articles, $\mathcal{D}$. Therefore, this solution may not be feasible when $\mathcal{D}$ is large. In this section, we consider practical ways to speed up the computation even more.

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

We then find the best chain 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 that generates a stream of chains, each one chosen from a larger set of candidate articles.

*Restricting transitions.* If every transition between two documents is possible, Algorithm `findOptimalChain` may evaluate $O(\mathcal{D}^K)$ chains in the worst case. However, many transitions are highly unlikely. In particular, if we guess that the value of the optimal chain is $\tau$, we can eliminate all transitions whose best possible coherence score is weaker than $\tau$. We use binary search to find $\tau$.

For this reason, we have introduced input parameter $G$. $G$ is a graph restricting the transitions that can participate in the chain. In the simplest case, $G$ is a directed clique; all transitions obeying chronological order are possible. Alternatively, we may remove all edges whose best possible coherence is below some threshold. In our experiments, we have chosen to restrict the degree of $G$ instead. For each $v \in V$, we

have used a technique similar to the random walks of Section 2.3 to determine $v$'s top $d$ neighbors, and included only those edges in $G$.

Restricting the transitions speeds up computation significantly. In addition, this approach also leads to a natural anytime algorithm: one can start from a sparse $G$ and incrementally add edges.

*Approximate solutions.* Many techniques have been proposed to speed up informed best-first strategies. For example, we may continue to expand a node even if its merit is not the highest in the queue. For example, if we keep expanding a node whenever it is at least $(1 - \alpha)$ times the best node we have seen so far (for some $\alpha \in (0, 1)$), we are guaranteed to end up with a $(1 - \alpha)$-approximation algorithm. This approximation is often faster in practice. In addition, our algorithm can be generalized into a bidirectional heuristic search.

## 4. EVALUATION

Performance evaluations of information retrieval tasks often focus 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, the number of articles mentioning Clinton was much higher than those mentioning Enron.

For each article, we extract named entities and noun phrases using Copernic Summarizer[2]. In addition, the NYT dataset includes important metadata, such as taxonomy and section. We remove infrequent named-entities and very common nouns and noun phrases (e.g., "year"). We then computed influence as described in Section 2.3. Note that the Copernic weights do not depend on the subset of articles; had we used another method, such as TF-IDF, the choice of corpus would affect the weights.

Our main 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 chains linking each pair using the following techniques.

— *Connecting-Dots.* As described in Shahaf and Guestrin [2010], but using the rounding technique we had at the time of the user studies, based on iteratively removing articles.[3]

  The typical value of $K$ was 6 or 7. $kTotal$ was set to 15, and $kTrans$ was set to 4. Picking parameters is still an open challenge, but we found held-out stories to be an effective method. We used the speed-up methods of Section 3.3, and allowed ten minutes for the creation of a chain.

[3]During each iteration, we solve the LP from Shahaf and Guestrin [2010]. 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 is guaranteed to stop after $|\mathcal{D}| - K + 1$ iterations. In practice, it reaches a solution within a few iterations.

— *Shortest-path.* We constructed a graph by connecting each document with its nearest neighbors, 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*[4]. GNT is a Web application that organizes news search results on a browsable, graphical timeline. The dataset is different, making comparisons harder. However, as GNT seems to include all of the New York Times articles, we believe its dataset is a superset of our corpus.
The input to GNT 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. The strings we used were "Clinton Lewinsky," "OJ Simpson," "Enron," "Daniel Pearl"+"World Trade Center," and "Lewinsky"+"Elections."
— *Event threading* TDT [Nallapati et al. 2004] is a method to discover subclusters in a news event and structure them by their dependency, generating a graph. We ran TDT on the same corpus, and 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.

Note that none of the other methods optimizes for a notion of coherence. Unfortunately, we could not find any tool that was designed to optimize over an objective similar to ours. The preceding methods were chosen since they represent the ways in which a contemporary news-reader might try to learn a news story.

First, we presented 18 users with a pair of source and target article. We gauged their familiarity with those articles, asking whether they believed 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 preceding methods in a double-blind fashion. We asked the users to indicate the following.

— *Relevance*. Which chain captures the events connecting the two articles better?
— *Coherence*. Which chain is more coherent?
— *Redundancy*. Which has more redundant articles?

By asking users to indicate coherence, we wanted to see whether the objective we propose captures the users' intuitive notion of coherence. Relevance and redundancy are two properties that we deemed essential for a good chain, despite not directly optimizing for them.

Helping users gain better understanding of a story is the main goal of this article. In order to quantify this, we also 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.

Example output chains are shown in Figure 9. Figure 10 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 top plot shows the effectiveness (closing the familiarity gap) for each of the methods. The bottom of the plot shows familiarity scores for each story before reading

[4]http://newstimeline.googlelabs.com

**Google News Timeline:**

- 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)

**Connect the Dots:**

- 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

Fig. 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, e.g., an article about a domain name that once belonged to bin Laden's family.

any chain. For example, we can see that the Enron story is the least-known story out of the five.

Our algorithm outperforms 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 by a single query, the effectiveness of GNT decreased.

Surprisingly, the performance of GNT for the OJ story was much worse than its performance for the Enron story. A closer examination revealed that the number of articles about OJ far exceeded the number of Enron articles. Many of the OJ articles were esoteric at best, so picking equally-spaced $K$ documents tended to produce poor results (a book of a former juror made it to the best-seller 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 not perform well. Event threading achieved better results; 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 bottom of Figure 10 shows the percentage of times each method was preferred to another in terms of relevance, coherence, and nonredundancy. Users could prefer one chain, or state that both are equally good/bad. Therefore, the numbers

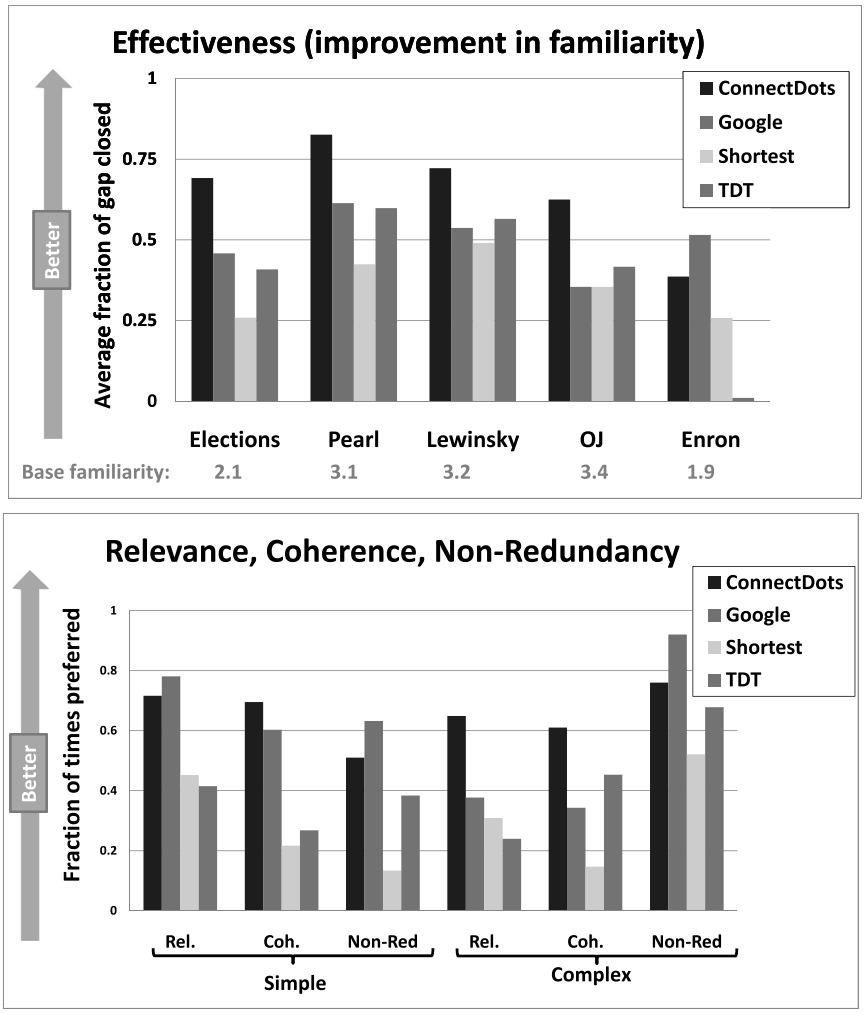



Fig. 10. Top: Evaulating effectiveness. The average (over users) of the fraction of familiarity gap, which was closed after reading a chain. Numbers at the bottom indicate the average familiarity with each story (on a scale of 1 to 5) before reading any chain. Bottom: Relevance, coherence, and nonredundancy (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 mark chains as "equally good," and therefore the numbers do not sum to 1. Our algorithm outperformed the competitors almost everywhere, especially for complex stories.

do not sum to 100%. The results are grouped based on the type of story (simple vs. complex). Our algorithm is among the best in all measures. Most importantly, it achieves the best coherence scores, especially for complex stories. We discuss some of the interesting findings in the following.

*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 does well in terms of relevance for simple stories. However, the chains it generates tend to include somewhat insignificant articles, especially for

- 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

Fig. 11. Chain refinement. The starred article was added in order to strengthen the last link.

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 for evaluating this work. Our algorithm outperforms the other methods, especially in the complex case. This indicates that the notion of coherence devised in this article 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 taking 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 2 (right). We provide the user with a mechanism to indicate areas in the chain that should be further refined; a refinement may consist of adding a new article, or replacing an article that 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.

In Figure 11, 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."

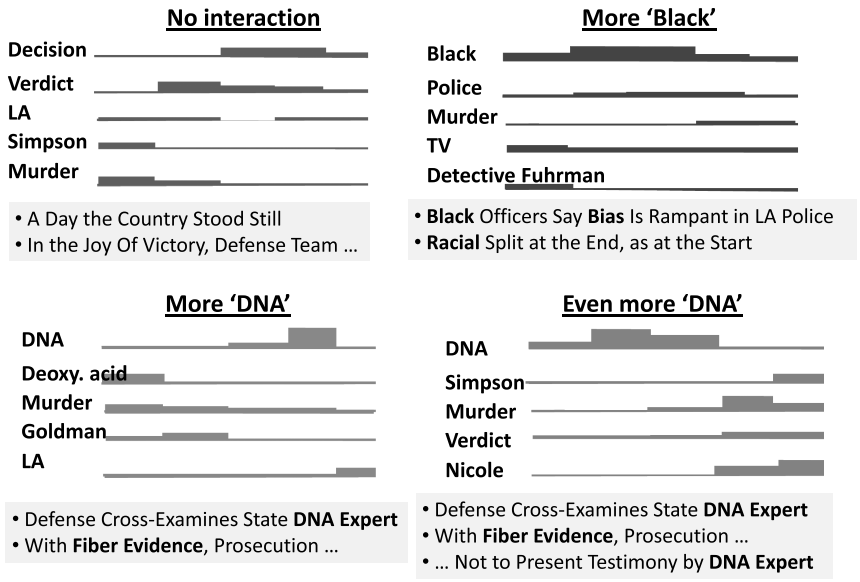



Fig. 12. A demonstration of our interactive component. The original chain is at the top left. The rest are derived from it by requesting the words "black," "DNA," and "DNA"×2. For each chain we show activation levels for the most important words, and a few selected articles.

In order to take the user's feedback into account, we augment our objective with importance weight $\pi_w$ for each word $w$.

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

$\pi_w$ are initialized to 1. When a user likes a word, its importance is increased by a multiplicative factor,

$$\pi_w = \pi_w \cdot (1 + \alpha).$$

When the user dislikes 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 cooccurrence information to distribute the effect among related words. For example, when a user clicks on "DNA," the words "blood" and "evidence" increase a little too. We use a method similar to the one in Section 2.3 to compute the cooccurrence score of word $w$, $\textit{co-occur}(w)$: using the same bipartite graph, $\textit{co-occur}(w)$ is the stationary distribution for random walks starting from the chosen word. Then, we update each $\pi_w$.

$$\pi_w = \pi_w \cdot (1 + \textit{co-occur}(w)) \cdot \alpha.$$

Figure 12 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 the user's feedback is as a constraint. Under this interpretation, the user imposes constraints on the accumulated influence of specific words in the chain. We keep a window for each word, representing the allowable range of total activation throughout the chain. The window is initialized to $[0, K-1]$: the word may not be active at all (zero total activation), or it may be fully active throughout the entire chain (total $K-1$ units). We update the window in response to user actions. Suppose that the user indicates that they want more of a word $w$, after seeing a chain in which $w$ had total activation $\rho$. This implies that the user wants the total activation of $w$ to be higher. If the current window for $w$ is $[c_{low}, c_{high}]$, we increase $c_{low}$.

$$c'_{low} = c_{high} - ((c_{high} - \rho) \cdot \alpha),$$

for some $\alpha \in (0, 1)$. If the user wants less of word $w$, we decrease $c_{high}$ in a similar way.

After updating the range window, we find the most coherent chain subject to the new constraints. If the user wanted more of word $w$, the resulting chain will demonstrate higher levels of influence for $w$, compared to the previous chain. 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 El-Arini et al. [2009] to find words that might be attributed to 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 users 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 3.3), obeying chronological order. We asked the users to indicate which chain better fit their requests. 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 could identify at least some of the words. Users identified at least one word 63.3% of the times.

In the future, we plan to test this component by allowing users to specify their own interests, and measuring their satisfaction. However, as the results of this future study are harder to compare across users, we chose to focus on experiments that will make the comparison clearer, although narrower.

## 6. CONNECTING ONE (OR LESS) DOTS

The system we have presented in the previous sections provides a simple mechanism to connect two fixed endpoints. However, in many real-life domains, this type of query is unnatural. Consider, for example, a researcher trying to understand the state of the art in some field. For him, there are no clear endpoints to connect. Even in our news domain, for which we developed the fixed-endpoint mechanism, this type of query

might be inappropriate. Users might not always know both the starting and ending points of a story. A more natural query mechanism allows users to fix a single article, and ask how the story developed from that point.
In this section, we consider the problem of forming a coherent chain when the endpoints are only partially specified.

Is this problem harder than our original problem? On one hand, the number of chains to consider is much larger. On the other hand, our algorithms seem easy to extend for the case of partially specified endpoints. If the start (end) point is unknown, we can add a virtual source (sink) node, connect it to all other nodes, and proceed as if the new node was a part of the input.

Let us demonstrate what happens when we extend our algorithm using virtual sources and sinks. Consider again the example from Figure 2. Instead of specifying both endpoints (Clinton's alleged affair and the 2000 election Florida recount), our input consists of the Clinton-Lewinsky article alone. The most coherent chain starting with this article is

- Talks over Ex-Intern's Testimony on Clinton Appear to Bog Down
- Is the Ex-Intern Getting Hostility or Sympathy?
- Lewisnky Would Take Lie Test in Exchange for Immunity Deal
- Calls to Intern from Clinton are the Envy of the Capital
- In America: The Clinton M.O.
- Lewinsky Ordered before Grand Jury on Ties to Clinton

As can be seen, the entire chain revolves around the Clinton-Lewinsky story. Furthermore, it is restricted to the very beginning of the story: in fact, the articles barely span more than a month's time.

Note that having two fixed endpoints served a purpose: it indicated possible directions that the user cares about. For example, specifying the Florida recount article forced the story to advance along the contemporaneous political events. Furthermore, when only a single endpoint is given, there is no incentive to ever change the topic, and the best chains would likely revolve around their starting point. Worse yet, when *no* endpoints are specified, this starting point might be uninteresting to the user.

In light of this, picking the most coherent chain no longer suffices. Instead, the key challenge seems to be balancing coherence and interest. It is not enough for a chain to be coherent; it should also cover topics that are important to the user.

How do we know which topics are important to the user? We start by having the user guide us as we build the chain. In the following we define an interactive variant of Connect-the-Dots, called `Connect-A-Dot`. In `Connect-A-Dot`, the user fixes a single article $d$, and incrementally builds a chain around it.

We focus on the case where $d$ is either the first or the last article in the chain (Where Do We Go From Here? and How Did We Get Here?, respectively). However, our techniques apply to the case $d$ is located at other positions as well.

### 6.1 Algorithm Overview

Intuitively, the algorithm is an interactive version of our search algorithm from Section 3, where the user guides the search. Our system starts from an input article and incrementally adds articles to the chain. At each step, the system computes the set of valid extensions to the current chain, which reach a certain coherence threshold. The valid extensions represent the many ways in which the chain can progress; we rely on user feedback to choose the next article. Unfortunately, the set of candidate articles is often too big to display. Therefore, our main challenge is to pick a small set of diverse candidate articles that cover all major aspects of the story.

**Algorithm 2:** `interactiveConnectDot`($G, d, isForward, b$)

**input** : $G = (V, E)$ graph, $d \in V$, *isForward* boolean indicating direction of time, $b$ number of alternatives to show the user.

```
// Initialize the chain
1 curChain = (d);
2 while true do
      // Find candidate articles
3     Cands= findExtensionsAboveThreshold(G, curChain, isForward) ;
      // Find a small subset which covers Cands well
4     A = maximizeCoverage(Cands, b) ;
      // Display subset to user, extend the chain by user's choice
5     v = getUserChoice(A) ;
6     curChain = extendChain(curChain, v, isForward) ;
```

For example, consider a user who is interested in the story leading to Guantanamo Bay closing. The user picks an input article and feeds it to the system. We then consider valid extensions of the input article. For example, some major aspects of the story (which we should cover) are Obama's promise, legal aspects, NGO reports, and suicide attempts.

Our algorithm's outline is described in Algorithm 2. At each iteration, our goal is to pick $k$ candidate articles and show them to the user. The articles should cover the different ways in which the chain may progress. In the future, we may enforce a minimum step size, to ensure that the chain is providing enough new information. As a first step, we calculate the set of documents Cands, which can serve as valid extensions to the chain we have built so far (Line 3). `findExtensionsAboveThreshold` evaluates all $O(|\mathcal{D}|)$ possible documents, and keeps the ones that are above a given threshold (either absolute or relative). Picking a good threshold is left for future work; in our experiments, we have found that keeping the top 10–20% provides breadth of topics without sacrificing coherence. Evaluating a single chain is done via the LP of Section 2.2, and takes very little time.

Next, we wish to pick a subset $\mathcal{A} \subseteq Cands$ of size $b$, which maximizes coverage of *Cands* (Line 4), displays them to the user, uses their choice to extend the chain (Lines 5–6), and reiterates. Line 4 lies at the heart of the algorithm. Before we can implement it, we first need to define coverage. Luckily, this is very similar to a problem we previously tackled in El-Arini et al. [2009], our goal was to pick a subset $|\mathcal{A}| = b$ of blog posts that covers the important stories in the blogosphere. Nevertheless, the same principles apply to any corpus of documents—and in particular, to the set of documents that could extend the chain. In the following, we briefly review those principles.

In order to define coverage, we first define the set of objects we are trying to cover. We refer to those objects as features. Features can be any arbitrary collection of objects, high- or low-level, e.g. significant words (such as named entities and noun phrases), topics extracted from the corpus, or even higher-level semantic relations. Documents are characterized by features: the relation between documents and features is captured by the covering function. More formally,

*Definition* 6.1 *Corpus*. A corpus is a triplet $\langle \mathcal{U}, \mathcal{D}, cover_\cdot(\cdot) \rangle$.
$\mathcal{U} = \{u_1, u_2, ...\}$ is a finite set of features, and $\mathcal{D}$ is a finite set of documents.
The relation between documents and features is captured by the *covering* function. For each $d_j \in \mathcal{D}$, $cover_j(\cdot) : \mathcal{U} \to \mathbb{R}^+$ quantifies the amount document $j$ covers feature $i$.

The coverage function $cover_.(\cdot)$ is a part of the input. In the simplest case, $cover_.(\cdot)$ is a binary indicator function, which effectively turns documents into subsets of features. In El-Arini et al. [2009], we explored softer notions of coverage functions, e.g., ones with probabilistic interpretations.

The coverage function is defined over single documents. However, we want to quantify the amount to which a set of candidate documents covers each feature $i$. For this reason, we wish to extend the coverage function to a function over sets of documents, $cover_{\mathcal{A}}(i)$. We view set-coverage as a sampling procedure: each document tries to cover feature $i$ with probability $cover_j(i)$. The feature is covered if at least one of the documents in the set $\mathcal{A}$ succeeded. Thus, as $\mathcal{A}$ grows, adding a document provides less and less extra coverage of feature $i$. Formally, we can define the probabilistic coverage of a feature by a set of documents $\mathcal{A}$ as

$$cover_{\mathcal{A}}(i) = 1 - \prod_{d_j \in \mathcal{A}} (1 - cover_j(i)). \tag{10}$$

We now know how much a set of documents covers each feature. However, some features are more important to cover than others. Next, we assign weights $w_i$ to each feature $u_i$, signifying the importance of the feature. For example, if features are words, the weights can correspond to their frequency in the dataset. Finally, we model probabilistic coverage of the entire corpus as

$$F(\mathcal{A}) = \sum_{i} w_i cover_{\mathcal{A}}(i). \tag{11}$$

Our goal now is to find the set of documents $\mathcal{A}$ that maximizes this objective function. Unfortunately, we can show by reduction from maximum coverage that this objective is NP-complete, suggesting that the exact maximization of this objective is intractable. Fortunately, our objective function satisfies an intuitive diminishing returns property: submodularity, which will allow us to find good approximations very efficiently. Although maximizing submodular functions is NP-hard [Khuller et al. 1999], we can take advantage of several efficient approximation algorithms with theoretical guarantees. In particular, we use the classic result of Nemhauser et al. [1978] which shows that the greedy algorithm obtains a $(1 - \frac{1}{e})$ approximation.

Now that we know how to pick a small set of documents that approximately maximizes coverage, we note that we can use the exact same method when no endpoints are specified (Connect-No-Dots). In this case, the user first selects a set of candidate documents (e.g., by performing keyword search), and we then use the greedy algorithm to propose a diverse set of candidate starting points. In the following, we demonstrate this technique.

### 6.2 Usage Example

We demonstrate our system on the New York Times dataset. We define our set of features as topics from a latent Dirichlet allocation (LDA) [Blei et al. 2003] topic model learned on the noun phrases and named entities described in the preceding. We can directly define $cover_j(i) = P(u_i \mid d_j)$, which in the setting of topic models is the probability that $d_j$ is about topic $i$. We use the Mallet [McCallum 2002] implementation of LDA with 20 topics and the default parameter settings. When we start from the general keyword "OJ Simpson," our top five suggestions for a starting point are the following.

(1a) A Bit Reluctantly, a Nation Succumbs to a Trial's Spell;
(1b) Few Can Avoid Harsh Glare Of Murder Trial's Spotlight;
(1c) Evidence Is Powerful, but He's Still O.J.;

(1d) The Simpson Factor in Race Relations;
(1e) Jurors and Judge Ito: Their Private Lives.

Those articles cover various aspects, from the media coverage, through evidence, race relations, and even judge and jury. When we pick (1c) evidence, the system proposes the following options.

(2a) Tempers Flare Over Simpson DNA Expert's Drug Use;
(2b) Blood Drops Analyzed for Simpson Jury;
(2c) Coroner Says Time of Death Is Imprecise;
(2d) Simpson Defense Changes Glove Tactics;
(2e) Former Simpson Juror Sees Weak State Case.

Most of the articles are related to evidence, but they cover different types of evidence, e.g., blood and time of death. When we pick (2d) glove, the system starts to converge. The articles are still mainly about evidence, and almost all of them mention the gloves.

(3a) Blood on a Simpson Glove Matches Victim's, Expert Says;
(3b) Near-Unanimous Verdict: Blunder at Simpson Trial;
(3c) Key Fibers Linked to Simpson Vehicle;
(3d) Simpson Defense Changes Glove Tactics;
(3e) North Carolina Judge Rules Against Simpson Legal Team.

Suppose that, in the previous step, we would have picked article (2a) DNA, instead of (2d) glove. In this case, the algorithm's choice of articles would be very different, and much more focused around DNA evidence:

(3a′) Simpson Prosecutors Decide Pathologist Won't Testify;
(3b′) Blood on a Simpson Glove Matches Victim's, Expert Says;
(3c′) Simpson DNA Papers Go to Smithsonian;
(3d′) Again Suggesting Botched Inquiry, Simpson Defense Cross-Examines State DNA Expert;
(3e′) At the Bar; The jury is still out on the effects of long, televised trials.

## 7. RELATED WORK

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

The narrative generation community [Niehaus and Young 2009; Turner 1994] has sought to explore the notion of narratives and ways to model them. What makes narrative different from a list of events? How can narratives be modeled? 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 [Rowe et al. 2009] may be useful for our purposes.

The search community has explored issues relating to our work. The Athens System [Vats and Skillicorn 2004] discovers novel information, whose existence is not suspected, given an initial set of keywords. The MMR [Carbonell and Goldstein 1998] metric aims at combining query relevance with information novelty, thus encouraging diversity. As opposed to our system, both methods assume a simple keyword query. Their output is also restricted to a list of documents (or clusters), with no attempt

at linking them into a coherent picture. However, it may be interesting to try and combine these methods with our Connect-A-Dot coverage notion (Section 6).

As for *event tracking*, some efforts have been made to classify news stories into broad categories using pattern matching and machine learning [Masand et al. 1992]. However, these methods assume the labels are known in advance, and thus are not applicable. Event detection [Kleinberg 2002; Yang et al. 1999] deals with discovering new events, but does not attempt to string different events together.

In contrast, email threading [Lewis and Knowles 1997] 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 work closest to ours, Nallapati et al. [2004] and Mei and Zhai [2005] studied how to discover subclusters 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—stories that can be summarized in one or two keywords ("tsunami," in Mei and Zhai [2005]). It is not clear how well this method does for more complex stories.

Other lines of research have explicitly proposed connecting two endpoints. Heath et al. [2010] propose building graphs, called Image Webs, to represent the connections between images in a collection, and discover meaningful paths in them. Images may depict the same static scene, or they may be related in a more subtle way (for example, two different buildings at a university are connected by a campus bus that frequently stops near each building). However, as the authors' main tool for path discovery was shortest-path, their chains may exhibit stream-of-consciousness behavior, similar to the shortest-path chains of Section 2.

Kumar et al. [2006] formulate a new data mining problem, called *storytelling*, as a generalization of redescription mining. Storytelling aims to explicitly relate object sets that are disjoint by finding a chain of approximate redescriptions between the sets. For example, if some London travel books ($Y$) overlap with books about places where popes are interred ($G$), and some of which are books about ancient codes ($R$), then the sequence of approximate redescriptions $Y \Leftrightarrow G \Leftrightarrow R$ is a story.

The strength of a story is determined by the weakest transition. The strength of a transition measured by its Jaccard coefficient (the ratio of the size of common elements to elements on either side of the redescription). Since this is a local measure, any global notion of coherence is lost again.

Several methods have concentrated on connecting two points of interest via multiple paths. In Faloutsos et al. [2004], the goal is to extract a small (amenable to visual inspection) subgraph that best captures the connections between two nodes of the graph. The authors interpret the graph as an electrical network, and choose the subgraph that can deliver as much electrical current as possible. In Hossain et al. [2010], the goal is to find hammock paths, which are a generalization of traditional paths. Both electrical networks and hammock paths offer some insight regarding the connection between their endpoints. However, compared to our chains, they have little structure.

In conclusion, our work differs from most previous work in several 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—to 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, or a graph of strong but local connections, our output is more structured: a chronological chain of articles maximizing a global coherence measure, and the flow

of influences along it. Often, visually exploring a system's results and interacting with it can reveal new and interesting phenomena.

## 8. CONCLUSIONS AND FUTURE WORK

In this article, 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 demonstrated 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.

Several avenues for future work readily suggest themselves.

*Representation.* Some stories cannot be fully represented as simple chains. In the future, we plan to explore richer forms of output, allowing for more complex tasks, e.g., creating a road map—a set of intersecting chains that covers a topic from several aspects.

*Semantics.* In this work, we have concentrated on syntactic features (in particular, words) as the fundamental building blocks of news stories. In fact, one of our goals was to explore whether simple, superficial features can be used to evaluate the coherence of a chain. In the future, we plan to introduce more high-level semantic features, extracted automatically [Mitchell et al. 2009] or manually.

In particular, it will be interesting to think of more qualitative notions of influence, and see how plugging them into our framework affects the resulting chains. For example, we expect semantic features to mitigate the importance of different writing styles. Our current technique may connect articles by the same reporter more often than expected, since reporters often have typical choices when synonyms are available; high-level features should reduce this problem.

*Context-specific chain length.* Some articles can be connected by a short chain, while others require a much-longer chain. However, in our current formulation, the user is required to specify the desired chain length $K$. Other formulations may be possible, e.g. designing an algorithm that will not take chain length as an input parameter.

*Document quality.* In this work we have used the New York Times corpus, where we regard all articles to be top-quality. Other corpora, however, feature articles of varying quality. In the future, we plan to take article quality into account when composing chains.

*Query characteristics.* We plan to explore the behavior of our system under different query characteristics. For example, in our experiments we only considered news stories that were associated with popular events. It would be interesting to test the approach for news stories that are less prominent in our news corpus.

In addition, we have only evaluated pairs of documents that are related. If the input documents are not related, our system will still find the most coherent chain between them, but that chain is likely to be bad. Indicating absolute chain quality is still an open question. In particular, the coherence of the chain does not provide a meaningful score to the user. In the future, we wish to find an absolute quality measure.

We believe that the system proposed in this article 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, such as e-learning,

intelligence, and scientific discoveries. The notion of coherence may be very different in different domains; for example, in biology, coherence might stem from causally-related molecular processes in a cell. Nevertheless, we expect that some notion of coherence will be fundamental to discovering connections in many domains.

In addition, we plan to apply our techniques to multiple domains at once, as significant scientific discoveries can come from forming connections between different fields. We believe that tools to automatically connect the dots can be a great vehicle to enable new discoveries.

## REFERENCES


BLEI, D. M., NG, A. Y., AND JORDAN, M. I. 2003. Latent Dirichlet allocation. *J. Mach. Learn. Res. 3*, 993–1022.

BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst. 30*, 107–117.

CARBONELL, J. AND GOLDSTEIN, J. 1998. The use of MMR, diversity-based re-ranking for re-ordering documents and producing summaries. In *Proceedings of the Annual ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

CHOUDHARY, R., MEHTA, S., BAGCHI, A., AND BALAKRISHNAN, R. 2008. Towards characterization of actor evolution and interactions in news corpora. In *Proceedings of the European Colloquium on IR Research (ECIR)*. 422–429.

DECHTER, R. AND PEARL, J. 1985. Generalized best-first search strategies and the optimality of A*. *J. ACM 32*, 505–536.

EL-ARINI, K., VEDA, G., SHAHAF, D., AND GUESTRIN, C. 2009. Turning down the noise in the blogosphere. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 289–298.

FALOUTSOS, C., MCCURLEY, K. S., AND TOMKINS, A. 2004. Fast discovery of connection subgraphs. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 118–127.

GABRILOVICH, E., DUMAIS, S., AND HORVITZ, E. 2004. Newsjunkie: Providing personalized news feeds via analysis of information novelty. In *Proceedings of the 13th International World Wide Web Conference (WWW)*. 482–490.

HEATH, K., GELF, N., OVSJANIKOV, M., AANJANEYA, M., AND GUIBAS, L. J. 2010. Image webs: Computing and exploiting connectivity in image collections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3432–3439.

HOSSAIN, M. S., NARAYAN, M., AND RAMAKRISHNAN, N. 2010. Efficiently discovering hammock paths from induced similarity networks. *Computing Research Repository*.

KEMPE, D., KLEINBERG, J., AND TARDOS, E. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 137–146.

KHULLER, S., MOSS, A., AND NAOR, J. S. 1999. The budgeted maximum coverage problem. *Inform. Process. Lett. 70*, 39–45.

KLEINBERG, J. M. 1999. Authoritative sources in a hyperlinked environment. *J. ACM 46*, 5, 604–632.

KLEINBERG, J. 2002. Bursty and hierarchical structure in streams. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

KUMAR, D., RAMAKRISHNAN, N., HELM, R. F., AND POTTS, M. 2006. Algorithms for storytelling. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 604–610.

LEWIS, D. D. AND KNOWLES, K. A. 1997. Threading electronic mail: A preliminary study. *Inform. Process. Manage. 33*, 2, 209–217.

MASAND, B., LINOFF, G., AND WALTZ, D. 1992. Classifying news stories using memory based reasoning. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. (SIGIR)*. 59–65.

MCCALLUM, A. K. 2002. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu.

MEI, Q. AND ZHAI, C. 2005. Discovering evolutionary theme patterns from text: An exploration of temporal text mining. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*. 98–207.

MITCHELL, T., BETTERIDGE, J., CARLSON, A., HRUSCHKA, E., AND WANG, R. 2009. Populating the semantic web by macro-reading Internet text. In *Proceedings of the Semantic Web Conference (ISWC)*. A. Bernstein, D. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta, and K. Thirunarayan Eds., Lecture Notes in Computer Science, vol. 5823, Springer Berlin, 998–1002.

NALLAPATI, R., FENG, A., PENG, F., AND ALLAN, J. 2004. Event threading within news topics. In *Proceedings of the ACM 13th International Conference on Information and Knowledge Management (CIKM)*. 446–453.

NEMHAUSER, G. L., WOLSEY, L. A., AND FISHER, M. L. 1978. An analysis of approximations for maximizing submodular set functions. *Math. Program. 14*, 265–294. 10.1007/BF01588971.

NIEHAUS, J. AND YOUNG, R. M. 2009. A computational model of inferencing in narrative. In *Proceedings of the National Conference on Artificial Intelligence, Spring Symposium (AAAI)*.

ROWE, J. P., MCQUIGGAN, W., ROBISON, J. L., MARCEY, D. R., AND LESTER, J. C. 2009. StoryEval: An empirical evaluation framework for narrative generation. In *Proceedings of the National Conference on Artificial Intelligence Spring Symposium (AAAI)*.

SHAHAF, D. AND 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)*. 623–632.

TURNER, S. R. 1994. *The Creative Process: A Computer Model of Storytelling and Creativity*. Lawrence Erlbaum Associates.

VATS, N. AND SKILLICORN, D. B. 2004. The Athens system for novel information discovery. Tech. rep. 2004-489, Queen's University School of Computing.

YANG, C. C., SHI, X., AND PING WEI, C. 2006. Tracing the event evolution of terror attacks from on-line news. In *Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI)*. 3975. 343–354.

YANG, Y., AULT, T., PIERCE, T., AND LATTIMER, C. W. 2000. Improving text categorization methods for event tracking. In *Proceedings of the 23th Annual International ACM SIGIR Conference on Research And Development in Information Retrieval (SIGIR)*. 65–72.

YANG, Y., CARBONELL, J., BROWN, R., PIERCE, T., ARCHIBALD, B. T., AND LIU, X. 1999. Learning approaches for detecting and tracking news events. *IEEE Intell. Syst. 14*, 32–43.

Received February 2011; revised July 2011; accepted July 2011