Unweaving a Web of Documents

R. Guha
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120.
guha@almaden.ibm.com

Ravi Kumar
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120.
ravi@almaden.ibm.com

D. Sivakumar
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120.
siva@almaden.ibm.com

Ravi Sundaram[*]
College of Computer and Information Science
Northeastern University
Boston, MA 02115.
koods@ccs.neu.edu

ABSTRACT

We develop an algorithmic framework to decompose a collection of time-stamped text documents into semantically coherent threads. Our formulation leads to a graph decomposition problem on directed acyclic graphs, for which we obtain three algorithms — an exact algorithm that is based on minimum cost flow and two more efficient algorithms based on maximum matching and dynamic programming that solve specific versions of the graph decomposition problem. Applications of our algorithms include superior summarization of news search results, improved browsing paradigms for large collections of text-intensive corpora, and integration of time-stamped documents from a variety of sources. Experimental results based on over 250,000 news articles from a major newspaper over a period of four years demonstrate that our algorithms efficiently identify robust threads of varying lengths and time-spans.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Miscellaneous*

General Terms

Algorithms, Experimentation, Measurements

Keywords

News threads, Graph algorithms, Graph decomposition

[*]Part of this work was done while the author was visiting the IBM Almaden Research center.

1. INTRODUCTION

The science of organizing and searching document collections for the purpose of perusal by human users is emerging into a mature discipline. The body of work in this field has evolved to a point where several basic paradigms — relevant elements of information retrieval, link analysis, synthesis of ranking functions, and clustering techniques — may be considered fairly well understood. On the other hand, the ubiquity of search as the single most important entry point into users' interaction with large repositories of information has brought to the fore a new class of questions for scientific inquiry. Some of the prominent issues include developing methodologies for searching and organizing specific collections of information — for example, enterprise data, e-commerce data, medical/legal libraries, news sources, desktop data, etc. Each of these focused problems has its unique set of characteristics in terms of scale, relevant data models, robustness requirements, and tolerance to spam, among other features.

In this paper, we provide an algorithmic framework for the analysis of large collections of *time-stamped* text documents. The primary goal of our framework is to decompose a collection of documents into semantically significant *threads*.

Before we state the specific technical aims of our work, let us pause and explore the space of potential applications, some of which will influence the choice of algorithmic models and ideas we employ, as well as help us delineate our approach from prior relevant work.

Motivation.

At a fairly broad level, we consider collections of documents of *text* in a single but arbitrary language; the only significant requirement we make is that each document have a unique *time-stamp*. We do not assume that the documents are drawn from a single source, nor do we assume that all documents in the collection pertain to a single theme. A few motivating examples follow.

Consider an archive of news articles from one or more fairly generic sources (e.g., news feed from AP or Reuters). A convenient means to organize, summarize, browse, and search such data would benefit greatly from the decomposition of the collection into threads of relevant articles. The notion of a thread is rather pervasive in the news media;

in fact, during the evolution of USENET news as a forum for exchange of ideas, an important step was the development of "threaded news readers." Surprisingly, the search facility at common sources of news articles (CNN, The New York Times, etc.) do not offer a threaded interface to their archives — "sort by date" and "sort by relevance" appear to be the most commonly available ranking options.

As an example, the query "Colin Powell" produces a few thousand responses, and neither of these sorting criteria is quite adequate, especially considering the fact that articles about Mr. Powell can be naturally organized into news threads corresponding to various time periods of his career (e.g., the Gulf war of 1990, his appointment as Secretary of State in 2001, his recent resignation, etc.). Furthermore, this organization is not unique, and depends on the "granularity" of the decomposition — a long thread often consists of several shorter threads that are loosely bound together. For example, a collection of articles describing preparations for a war on Iraq emerges as a natural sub-thread of the longer thread during his tenure as Secretary of State, and within this, a European tour by Colin Powell emerges as a finer sub-thread.

A somewhat more interesting example in this genre is the case of news articles from a variety of media that match a given keyword (e.g., IBM). These have a natural temporal flavor; they also admit natural partitioning not just in terms of the news source, but also in terms of the emphasis of the content. For example, articles about IBM can be divided into business articles about the performance of the IBM stock, articles containing reviews and descriptions of IBM products, etc.

A second motivating example stems from medical and legal case histories that practitioners of these fields routinely consult. While the importance of semantic coherence is fairly obvious, the role of temporal coherence requires a few words of explanation. In fields that are constantly undergoing evolution (as law and medicine certainly are), it is useful to access information sequenced meaningfully along the time axis; for example, in medicine, the identical set of symptoms has led, over time, to rather different conclusions about potential causes and treatments. In such cases, it is not enough to produce (in response to a query that consists of the symptoms) all "relevant" prior cases; it is equally important to order them temporally and find logical "break points" in the sequence at which the various threads of analyses (diagnosis, treatment, etc.) can be divided. The notion of threads is fairly natural in scientific literature and patent databases as well.

Our framework.

We propose a novel algorithmic framework that captures the essence of the thread decomposition problem. One of the highlights of our approach is a crisp combinatorial modeling of the problem as a graph decomposition problem for directed acyclic graphs. Loosely speaking, the graph problem is to decompose a given directed acyclic graph into as few node-disjoint paths as possible while ensuring that as many nodes as possible participate in at least one path. While this formulation is not particularly surprising, the main advantage of our rendition as a graph decomposition problem is the fact that the "modeling" and "computational" aspects of the thread extraction problem are decoupled — one may plug and play algorithms for either part of the problem depending on the characteristics of the instances at hand and/or the computational capabilities and limitations of the available resources.

The graph decomposition problem we develop in this paper exhibits, as many graph problems do, an amazing spectrum in terms of its computational complexity, from NP-hard incarnations to ones that admit efficient algorithms — not just polynomial-time efficiency but efficient in the stream–sort model of computation. We show that the class of instances we are interested in — directed and acyclic graphs — admits polynomial-time algorithms for the graph decomposition problem; this algorithm is based on minimum cost flow. We also propose a more efficient two-pronged algorithmic attack for two special cases of the problem, one based on maximum matching in bipartite graphs, and the other based on dynamic programming. Each approach has its own advantages, depending on the scale of the problem instances. The matching-based algorithm needs sufficient memory to hold the graph and so is applicable only if the underlying graph is small. On the positive side, matching is a well-understood graph algorithmic primitive, which allows us relatively easy use of (publicly) available and fairly well-optimized code. The dynamic programming approach we develop is tailored to be efficient in the stream–sort model [1], therefore admits implementations that require substantially less memory than would be required to hold the graph.

The work of [1] places the spotlight on *streaming* and *sorting* as general computational primitives. Indeed, our dynamic programming algorithm highlights the importance of developing stream–sort algorithms for problems on directed and acyclic graphs in the context of search, organization, and mining of document collections.

A remark on relationship of our work to clustering. Threads in temporally ordered corpora offer a clean notion of clusters, and it is natural to ask if one may simply cluster the documents according to standard clustering algorithms, and order the documents within each cluster according to their timestamps. While this is possible in principle, in practice, one needs to examine all pairs of documents to determine their similarity or dissimilarity; in the special case of temporally ordered document collections, one may take advantage of the sparsity of the term–document relation and devise algorithms where the document–document similarity relation has on average $O(nt)$ entries, where n is the number of documents and t is the average thread length. Our algorithms take special advantage of the temporal structure of documents and indices, and work with relations with $O(n)$ entries. This is not only more compact, but it has the advantage that the clustering algorithms (in our case, graph algorithms) run much more efficiently. For example, if a collection of n documents with roughly $\sqrt{n}$ threads of length $\sqrt{n}$, the difference in the size of the relation is a factor of $\sqrt{n}$, a significant savings in practice.

A second advantage of our approach compared to traditional clustering is that news threads often have a natural drift of topic over time and our algorithms gracefully allow for this, whereas in traditional clustering methods a primary objective is to keep each cluster tightly focused.

Experimental results.

We illustrate the performance of our algorithms by presenting experimental results from a fairly large corpus of news articles (roughly 250,000 news articles that span a time period of over four years). Our experiments were conducted on the corpus of all articles, as well as on subcollections that contain specific query terms. These experiments indicate the feasibility of our algorithms at fairly large scale, and also demonstrate the efficacy of our framework in allowing the identification of several natural threads of news stories, some that span several months, and some that have rather large "dormant gaps" in the middle, which makes the task algorithmically quite challenging.

2. RELATED WORK

The related work falls into the following categories: topic detection and tracking, analysis of news and news events, automatic construction of hypertext and hyperlinks, and clustering/automatic identification of communities.

Topic detection and tracking (TDT) refers to automatic techniques for discovering, threading, and retrieving topically related material in streams of data. The literature on TDT is extensive and we review only a few (see, for instance, `http://www.lt-world.org/elf/elf/collatequery?h_search=technology&c_technology_name=Topic+Detection&exakt=ja`). Clustering and text retrieval techniques were used by Yang et al. [26, 5] to automatically detect novel events from a temporally-ordered sequence of news stories. Statistical methods were used by Swan and Allan [23] to automatically generate an interactive timeline displaying major events in a corpus. Most of the work in TDT is focused on detecting new and significant events in a timeline; the usual tools are clustering and machine learning. We, on the other hand, are interested in detecting not just the novel/major events, but all temporally and semantically connected set of events. Automatically identifying threads in document collections is also related to burst analysis and event analysis. Kleinberg [17] models the generation of bursts by a two-state automaton and proceeds to automatically detecting bursts in sequence of events; he looks for the burst of a single keyword. We, however, are interested in enumerating all threads and using all terms.

News articles and news groups have been analyzed in the context of search and data mining. Agrawal et al. [2] study the use of link-based and graph-theoretic methods to partition authors into opposite camps within a given topic in the context of newsgroups. Finding news articles on the web that are relevant to news currently being broadcast was explored by Henzinger et al. [15]. Uramoto and Takeda [25] describe methods for relating multiple newspaper articles based on a graph constructed from the similarity matrix. Allan et al. [6] propose several methods for constructing one-sentence temporal summaries of news stories. Smith [22] examines collocations of dates/place names to detect events in a digital library of historical documents.

There has been lot of work on automatically generating hypertext, beginning with the thesis of Allan [4]. The work most closely related to ours is that of Dalamagas and Dunlop [9] and Dalamagas [8]. They consider the problem of automatic creation of hyperlinks for news hypertext that is tailored to the domain of newspaper archives. Their method is based on traditional clustering tools. Dalamagas [8] also explores the use of elementary graph-theoretic tools such as connected components to identify threads in news articles and builds a prototype system. Smeaton and Morrissey [21] use standard information retrieval techniques to compute a graph that is based both on node-node similarity and overall layout of the hypertext; they then use this graph to automatically create hyperlinks. Blustein [7] explore the problem of automatically creating hyperlinks between journal articles. Green [14] develops a notion of semantic relatedness to generate hypertext. Most of these work, however, focus on examining the text of a news article and adding hyperlinks to other news articles based on terms, dates, events, people, etc; they do not address the problem of identifying threads in news collection and do not fully use the temporal nature of the underlying data.

Our problem is also related to, yet different from, the problem of identifying communities in large graphs. Community identification has been studied extensively in the context of web pages, web sites, and search results. Trawling refers to the process of automatically enumerating communities from a crawl of the web, where a community is defined to be a dense bipartite subgraph; an algorithm to do trawling via pruning and the *a priori* algorithm [3] was presented in [19]. A network flow approach to identifying web communities was given by Flake et al. [11]. Local search methods were used to identify communities satisfying certain special properties in the work of [18]; their interest was to extract storylines from search results. Like clustering, none of these algorithms, uses a temporal ordering of the documents.

3. ALGORITHMS FOR FINDING THREADS

3.1 Preliminaries

A *directed graph* $G = (V, E)$ consists of a set V of nodes and a set $E \subset V \times V$ of edges. A graph is *weighted* if there is a weighting function $w : E \rightarrow R^+$. A *directed path* p from u to v is a sequence of nodes $u = w_1, \ldots, w_k = v$ such that $(w_i, w_{i+1}) \in E$ for every $i = 1, \ldots, k-1$; it will be clear from the context whether we refer to the nodes or edges in a path. A directed cycle is a non-trivial directed path from u to itself. A *directed acyclic graph* is a directed graph with no directed cycles.

3.2 The relevance graph

We are given a collection of documents $\mathcal{D} = \{D_1, \ldots, \}$ and the goal is to build a "relevance" graph that will help us identify the threads in the collection. Let $t(D)$ give the timestamp of a document. For simplicity, we will assume that the documents can be linearly-ordered according to their timestamps with D_1 being the earliest document, i.e., $i < j \Rightarrow t(D_i) < t(D_j)$. The goal of the graph is to express the relationship/relevance between two documents.

A natural way to construct a graph would be take every pair of documents $D, D' \in \mathcal{D}$ and add the edge (D, D') if and only if D and D' are deemed related. (We adopt the convention that whenever we add an edge between two documents D and D', it is always directed from D to D' if $t(D) < t(D')$ and from D' to D if otherwise.) Unfortunately, this approach is not scalable. Even if $|\mathcal{D}|$ is only of the order of thousands, it is not possible to construct or represent this graph without compromising on efficiency and scalability.

Therefore, we take an entirely different approach to construct a relevance graph out of the documents. Let $\mathcal{T} = \{T_1, \ldots\}$ be the "terms" in the document collection. The set of terms is chosen by some appropriate text parsing or entity extraction mechanism and depends on the particular application domain; in this section, we will not address the issue of how the terms are chosen. Suppose we build the term–document matrix M that corresponds to the terms and documents. Here, $M(D, T)$ is the amount of "presence" of the term T in the document D; once again, we will not address the issue of how M is constructed in this section (see Section 4). Our purpose is just to use this matrix M to build the relevance graph.

We illustrate the construction of the relevance graph in the case when M is binary, i.e., $M(D, T) = 1$ if and only if the term T occurs in the document D. For each term $T \in \mathcal{T}$, we consider $D_T = \{D'_1, \ldots\}$, the set of document postings corresponding to the term, in order of their timestamps. Let w be a window parameter. We add the edge (D'_i, D'_j) if and only if $|i - j| \leq w$, i.e., we add an edge between two documents if they are at most w apart in their ordering in D_T; note that it could very well be the case that $|t(D'_i) - t(D'_j)| \gg w$. At the end of this step, we end up with a multigraph on documents. We use a simple threshold parameter τ to "round" the multigraph to obtain the relevance graph.

Algorithm Build-relevance-graph $(\mathcal{D}, \mathcal{T}, w, \tau)$

```
Let $M$ be the term–document matrix
For $T \in \mathcal{T}$ do
Let $D_T = \{D'_1, D'_2, \ldots\}$
    For $D'_i, D'_j \in D_T$ do
        If $|i - j| < w$ then $E_{D'_i, D'_j} = E_{D'_i, D'_j} + 1$
For every nonzero entry $E_{D_i, D_j}$ do
    If $E_{D_i, D_j} \leq \tau$ then $E_{D_i, D_j} = 0$
    If $E_{D_i, D_j} > \tau$ then $E_{D_i, D_j} = 1$
Return nonzero entries of $E$
```

There are several merits to our way of constructing the relevance graph. Firstly, we avoid the quadratic blow up in time and space by restricting our attention to a small number of document pairs. At the same time, we do not compromise on documents that are related to each other but do not occur close to each other in time. The window parameter w lets us control this behavior. Secondly, using the threshold τ lets us have additional control in terms of rounding. By picking a suitably large threshold, documents that are brought together by small number of spurious co-occurring terms can be separated. Thirdly, it is an easy modification of our algorithm to deal with a non-binary term–document matrix, with additional information such as td–idf; see Section 4. Fourthly, the construction can also be modified to work with a documents that not totally ordered, but just bucket-ordered. Finally, the construction of this graph for the entire document collection is a one-time step and hence does not contribute to the run-time cost.

Since we are interested in identifying threads, it suffices to work with the connected components of the relevance graph, treated the edges as undirected. To find connected components in this graph, we resort to the classical union-find algorithm. This algorithm maintains a family of sets via a standard union-find data structure. Initially, the family contains only singleton sets, each consisting of exactly one node of the graph. For every edge (u, v), the sets in the family containing u and v are merged. It is easy to see that after all the edges are processed, each set in the family contains the nodes of a connected component. For a simple algorithm for connected components in the stream–sort model, see Aggarwal et al. [1]. For the remainder of this section, we will find the threads in a single connected component.

3.3 The (a, b)-threads problem

Informally, threads represent directed paths in the relevance graph. Given such a graph, the problem of finding the best threads can be naturally cast in a combinatorial optimization framework. While there are several ways to carry out this, we adopt the following formulation. Given a directed graph, find small number of disjoint directed paths to cover as many nodes as possible in the graph. Formally, the decision version of the (a, b)-threads problem is stated as:

PROBLEM 1 ((a, b)-THREADS PROBLEM). *Given a directed graph $G = (V, E)$ and parameters $a, b > 0$, are there at most a node-disjoint directed paths such that the number of nodes included in the union of the paths is at least b?*

In this most general bicriteria formulation, the (a, b)-threads problem is as hard as the Hamiltonian path problem in directed graphs [12]. If $a = 1$, then maximizing b is the longest path problem.

FACT 2. *$(1, n)$-threads problem for directed graphs is NP-hard.*

3.4 An exact solution for directed acyclic graphs

We now solve the (a, b)-threads problem exactly for directed acyclic graphs. The solution is based on solving a minimum cost flow problem, for a which a polynomial time algorithm is known To recap, in a *minimum cost flow* problem, we are given a graph with both capacities and non-negative costs on edges and the goal is to push as much flow as possible from source to sink, while minimizing the total cost of the flow. Here, the cost is defined to be the sum over all edges, the product the edge cost and the flow through the edge. This problem can be solved in polynomial time (see, for instance, [10, 24, 13]); typically, the running time is $\tilde{O}(nm)$, where m is the number of edges. If all the costs and capacities are integral, there is an optimal integral flow as well. Furthermore, the problem can be solved in polynomial time even if some of the costs are negative, as long as the graph is acyclic.

In the (a, b)-threads problem, recall that we are given a directed acyclic graph $G = (V, E)$ and the goal is decide if there are a threads in the graph such that the number of nodes covered is b. From G, we first construct a graph $G' = (V', E')$ as follows.

$$V' = \{v', v'' \mid v \in V\} \ \cup \ \{s, t\}$$

and

$$\begin{aligned} E' \ &= \ \{(v', v'') \mid v \in V\} \\ &\cup \ \{(u'', v') \mid (u, v) \in E\} \\ &\cup \ \{(s, v') \mid v \in V\} \ \cup \ \{(v'', t) \mid v \in V\} \end{aligned}$$

All the edges of the form $(v', v''), v \in V$ are assigned cost -1 and the remaining edges are assigned cost 0. All the edges in G' have unit capacity. See Figure 1 for an illustration.

Now, we run the minimum cost flow algorithm on G' to see if a units of flow can be pushed from source s to sink t. Since G' has unit edge capacities, an integral flow exists. G is acyclic and our construction ensures that G' is acyclic as well. Therefore, the algorithm can still solve the problem on G' even though some of its edges have negative costs.

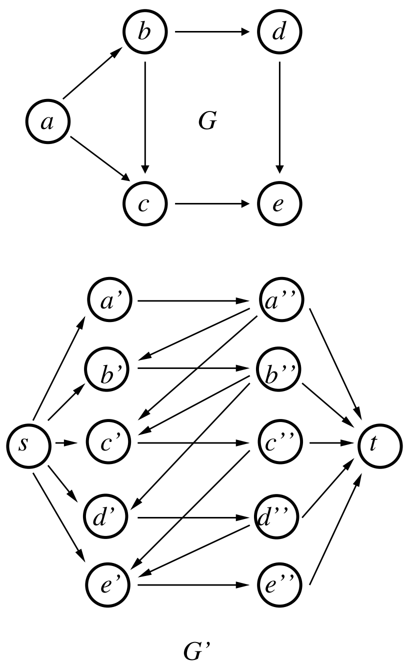

Figure 1: Illustration of the minimum cost flow-based algorithm.

From the construction, note that being able to route a units of flow from s to t means that there are a edge-disjoint paths in G'. Since the cost of each edge is -1, the minimum cost flow algorithm chooses to route the flow through as many edges of the form (v', v'') as possible. Now, since each node v in G is split into v' and v'', edge-disjoint paths in G' correspond to node-disjoint paths in G. Thus, each of the a unit flow path from s to t in G' corresponds to a node-disjoint path in G and if $-b$ is the value of the minimum cost flow, then the total number of nodes participating in these paths is b.

```
Algorithm Mincost-flow-based-threads (G)
  Construct $G' = (V', E')$ from $G = (V, E)$
  $F$ = MinCostFlow ($G'$)
  For each unit flow $s \to u' \to u'' \cdots w' \to w'' \to t$ in $F$
      Output thread $u \to \cdots \to w$ in $G$
```

Theorem 3. *The above algorithm solves the (a, b)-threads problem for directed acyclic graphs in polynomial time.*

The minimum cost formulation is quite powerful and has several advantages. Firstly, it is easy to persuade the algorithm to look only for threads with a minimum length ℓ. This can be accomplished by setting the cost of the edges (v'', t) to be $+\ell$. Secondly, the formulation permits the dual version of the problem to be solved: given the number of documents b, minimize the number of threads a to cover all the b documents. The solution follows from the monotonicity of the (a, b)-threads problem, i.e., if (a, b)-threads is feasible, then (a', b)-threads is also feasible for all $a' \geq a$.

Finding the minimum cost flow, even though can be done in polynomial time, is prohibitively slow if the graph is moderately large. In the next two sections, we provide very simple algorithms that permit more efficient implementations but solve weaker versions of the (a, b)-threads problem. The first algorithm is based on bipartite matching and the second algorithm is based on dynamic programming.

3.5 A matching-based algorithm

We note a simple lower bound which is the inspiration behind the matching-based algorithm.

Lemma 4. *$b \geq M(G)$, where $M(G)$ is the size of the maximum matching in G, when treated as an undirected graph.*

Using this, the first realization is that if we appropriately relax the bicriterion formulation of the (a, b)-threads problem, then the problem becomes simpler. The relaxation is to ignore a and just try to maximize b. Then, the problem can be solved optimally via the maximum matching algorithm. The only non-trivial aspect is that the matching has to be performed on a slightly different graph, whose construction is given below.

Recall that in the *bipartite maximum matching* problem, we are given an undirected bipartite graph $G = (U, V, E)$ with $|U| = n = |V|$ and $E \subseteq U \times V$. The goal is to find a subset of edges $E' \subset E$ of maximum cardinality such that each node in U, V is incident on at most one of the edges in E'. Maximum matching in bipartite graphs can be solved in $O(n^{5/2})$ time via the algorithm of Hopcroft and Karp [16] and can be approximated in near-linear time to within a factor of 2 via maximal matching.

Given a directed acyclic graph $G = (V, E)$, we construct an undirected bipartite $G' = (V', V'', E')$. The node set $V' = \{a' \mid a \in V\}$ $V' = \{a'' \mid a \in V\}$, i.e., consists of a', a'' for every $a \in V$. The edge set is given by

$$E' = \{(u', v'') \mid (u, v) \in E\}.$$

Figure 2 shows an example construction. Intuitively, selecting an edge $(u', v'') \in E'$ represents selecting the directed edge $(u, v) \in E$ and the matching constraint ensures that each node in G is matched at most once. After finding the maximum matching, the edges in the matching are referred back to the edges in G to construct the directed paths in G. We use the standard union-find data structure to recover the threads from the matched edges.

```
Algorithm Matching-based-threads (G)
  Construct $G' = (V', V'', E')$ from $G = (V, E)$
  $M'$ = MaxMatching ($G'$)
  For each $(u', v'') \in M'$ do
      Union (Find ($u'$), Find ($v''$))
```

It is easy to see that a collection of threads in G of size b naturally implies a matching of size b in G'. Conversely, any matching of size b in G' can be decoded into a collection of node-disjoint paths in G; the matching criterion translates to the node-disjointness requirement. Thus,

Lemma 5. *The above algorithm solves the $(*, b)$-threads problem in polynomial time.*

While the matching-based formulation has the elegant appeal of solving a special case of the (a, b)-threads problem exactly, it suffers from the following two serious shortcomings. Firstly, the best implementations of maximum matching in bipartite graphs take $O(n^{5/2})$ time. This algorithm loads

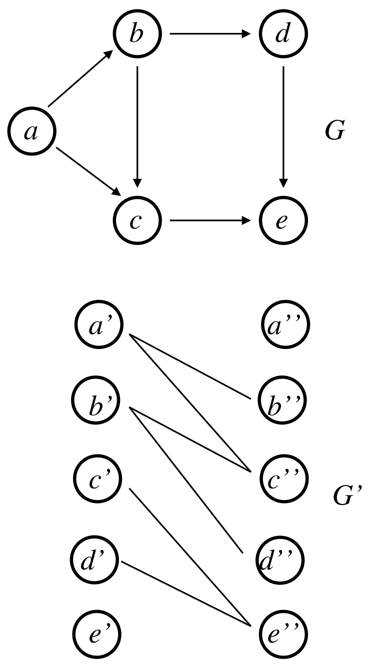

Figure 2: Illustration of the matching-based algorithm.

the entire graph into memory and hence becomes impractical for especially large graphs. Furthermore, finding maximum matching in the stream–sort model is a well-known open problem. Secondly, since matching maximizes b, it might produce threads that are fragmented. For instance, Figure 3 shows an instance where the matching-based algorithm might identify several small threads while there is a clear "good" thread. Note that this thread will indeed be discovered by the minimum cost flow algorithm with $a = 1$.

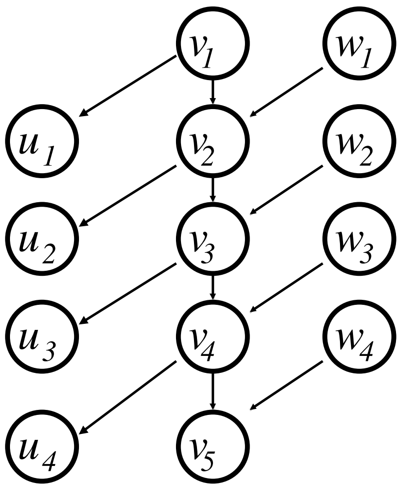

Figure 3: A case where matching identifies five separate threads $v_1 \to u_1$, $w_1 \to v_2 \to u_2$, ..., $w_4 \to v_5$ whereas the "best" thread to output is perhaps $v_1 \to \cdots \to v_5$.

In the next section we present a simple dynamic programming-based algorithm for a different relaxation of the the threads problem. The main feature of this algorithm is its realizability in the stream–sort model of computation.

3.6 A dynamic programming-based algorithm

In this section, we will present two algorithms for extracting threads from a collection of documents represented as a graph obtained in Section 3.2 based on the dynamic programming paradigm. The first algorithm of this section repeatedly removing long paths identified using dynamic programming. Here, we use the fact that if the graph is acyclic, then the longest path can be found very efficiently (unlike the general case).

Given the directed acyclic graph $G = (V, E)$ of documents, for $j \in V$, let $\ell(j)$ denote the longest path ending in node j of G. It is easy to see that given $\ell(i)$ for all $i < j$, where $<$ is an arbitrarily fixed topological order of G, we may recursively compute $\ell(j)$ as $\max_{i:(i,j)\in E}(1+\ell(i))$. This can be accomplished by a dynamic programming implementation with $O(n)$ nodes, where $n = |V|$. Furthermore, the dynamic programming algorithm for this problem admits a simple implementation in the stream–sort model if for $j \in V$ and all in-neighbors i of j, all the in-edges of i appear before all the in-edges of j, that is, the edges are ordered topologically by their destination node.

Furthermore, after removing the nodes that are included in a longest path, we may repeat the algorithm with the remaining edges to obtain further paths. Using an efficient hash table to keep track of covered nodes, we obtain a very efficient implementation of this algorithm.

Algorithm DP1-based-threads $(G = (V, E))$

```
Repeat
    $p$ = longest path in $G$
    Output $p$
    $G = G\backslash p$
Until $G$ is empty
```

Unfortunately, the above algorithm does solve any version of the (a, b)-threads problem. Below, we propose a more sophisticated dynamic programming algorithm to solve the (a, b)-threads problem where b is fixed to be n and the goal is to minimize a. While this dynamic program computes the maximal chain cover of the directed acyclic graph, the crucial point is that it can be implementable in the stream–sort model.

Given the directed acyclic graph $G = (V, E)$ of documents, and an integer parameter T denoting the maximum number of allowed threads, the goal is to find out if the nodes of V can be covered with at most T node-disjoint directed paths. Let τ denote an arbitrarily fixed topological order on the nodes of G; wlog. we may assume that the set $\{\tau(i) \mid i \in V\}$ is exactly the set $\{1, \ldots, n\}$, where $n = |V|$, so that we may identify the nodes with their rank in τ. Define the table $\textsc{cover}(k, t, i) = 1$ if all nodes $\le k$ can be covered with at most t node-disjoint paths such that node i is a leaf in one of these paths. For $j \in V$, suppose we have $\textsc{cover}(j-1, t, i)$ available for all i such that $(i, j) \in E$; we will show how to determine the entries $\textsc{cover}(j, t, i)$ for all $i \le j$. Namely, $\textsc{cover}(j, t, j) = 1$ if and only if for $\textsc{cover}(j-1, t, i) = 1$ for some i such that $(i, j) \in E$. Furthermore, $\textsc{cover}(j, t, i) = 1$ for $i < j$ if and only if $\textsc{cover}(j-1, t, i)$ and there is some node i' such that $i' \ne i$ and $\textsc{cover}(j-1, t, i') = 1$ and $(i', j) \in E$. Finally, $\textsc{cover}(j, t+1, j) = 1$ if $\textsc{cover}(j-1, t, i)$

is true for some $i < j$, and similarly $\text{COVER}(j, t+1, i) = 1$ if $\text{COVER}(j-1, t, i) = 1$.

These recurrences lead to an algorithm in the stream–sort model where we maintain a $T \times n$ table corresponding to the second and third co-ordinates of the table COVER — note that if the edges are presented in topological order of the destination node, then we do not need a 3-dimensional array as suggested by the recurrence. A crucial implementation detail is that we do not have to compute $\text{COVER}(j, t, i)$ for every value of i (which would take n^2 steps from the definition); since we obtain all edges pointing into j at the same time, we only need to consider $N(j)^2$ pairs (i, i') that are relevant. To trace the actual paths, rather than resort to a 3-dimensional array of size $n \times T \times n$, we could compute $\text{COVER}(j, t, \cdot)$ and $\text{COVER}(j, t+1, \cdot)$ and *write* an output stream that contains information necessary to trace the paths. For example, if we determine that $\text{COVER}(j, t, j) = 1$ because for some i, $\text{COVER}(j, t, i) = 1$ and $(i, j) \in E$ (which we would at the time the edges with destination j are processed), we will write into the output stream the tuple (j, t, j, i). These tuples can be post-processed (again, using the sorting primitive) to construct the actual paths.

LEMMA 6. *The above algorithm solves the (a, n)-threads problem for directed acyclic graphs in polynomial time.*

4. EXPERIMENTS AND RESULTS

In this section we describe our experiments on news articles and the results we obtain.

4.1 Experiments

Data source.

Our data source is *The Hindu*, a daily newspaper from India. This popular newspaper is published in the English language, and is very broad in its coverage of news items, ranging from local to international articles of importance. We obtained all the articles in the online version of the newspaper (`www.hinduonnet.com`) from Jan 1, 2000 to Mar 31, 2004.

Preprocessing.

We obtained all the articles and parsed them to collect a subset of terms in each document. This step was based on an algorithm in [18] to extract the most important terms in a given document. Each news article comes with an obvious time stamp and we randomly break ties to obtain a linear ordering of the articles. A binary unweighted term–document relation is then constructed from the articles and the terms. Our index consists of approximately 29.5 million term–document pairs from a corpus of roughly 250,000 articles and 375,000 terms. Thus, on average, a term occurs in roughly 78 documents and a document contains around 118 terms.

Let $\mathcal{D}$ denote the collection of documents in our corpus, let $\mathcal{T}$ denote the collection of terms, and let $\mathcal{R}$ denote the term–document relation synthesized. The primary interface with the index is a query engine that takes a term T and produces the partial index $\mathcal{R}' \subseteq \mathcal{R}$ defined by $\mathcal{R}' = \{(T', D') \mid (T, D') \in \mathcal{R}\}$; that is, $\mathcal{R}'$ consists of the projection of $\mathcal{R}$ to the set of articles that contain T. Producing $\mathcal{R}' = \mathcal{R}'(T)$ from the term t takes advantage of the index sorted according each of the keys, terms and documents, and the output of this step is sorted by term. This enables the application of the relevance graph construction step described in Section 3.2. We then apply the connected components algorithm to decompose the relevance graph, and apply our thread-detection algorithms to each of the large connected components.

4.2 Results — Anectodal examples

We present our results in the form of interesting threads uncovered by our algorithms. For definiteness, we set the window parameter $w = 10$ and the threshold parameter $\tau = 5$ in all these examples, and employed the mincost-flow based algorithm. The graphs in these case were quite sparse, and, on average, consisted of about 2,500 edges. The entire process, from issuing the query to web page creation, takes a few seconds (unoptimized). Based on preliminary studies, the algorithm based on maximum matching is somewhat more efficient on larger graphs, and the one based on dynamic programming offered a complete partition of all documents that respond to a query. Qualitatively, the mincost-flow based algorithm appears quite robust and produces threads of varying lengths, while the matching based heuristic and the dynamic programming formulation tend to produce more short threads (since they maximize the number of documents covered).

The discussion below outlines various threads produced for some quries, and presents some of the more colorful examples. Complete listing of all threads for a few tens of queries (for the setting $w = 10$ and $\tau = 5$ with the mincost-flow based algorithm) are presented at `http://www.almaden.ibm.com/cs/people/siva/threads.html`, which the reader is invited to visit.

First, we consider the query 'Clinton'. This query produced about 30 threads of length 5 or more, and corresponded to several natural news stories involving the former U.S. President as well as the present U.S. Senator. This included threads that corresponded to various visits to India by Clinton, a visit by the Indian National Security Adviser to the U.S., a thread corresponding to the 2000 U.S. elections, threads related to Clinton's involvement in the conflicts in Ireland, Israel. In addition, the algorithm produced a thread on a local sports personality in India. Tables 1 and 2 present two sample threads, one that follows the thread of IRA attacks, and one that follows the drama surrounding the child Elian Gonzalez; note that the former thread spans several months, while the latter is much more restricted to a few weeks.

Next, we present some threads obtained from the graph corresponding the term 'terrorist'. This query produced various threads related to the terrorist attacks on September 11, 2001, in New York City, including specific storylines on the impact of the attacks on financial markets, world sports events, various editorials on how to proceed in the war against terror, the U.S. war in Afghanistan following the terror attacks, threads on various terrorism-related laws within India, an unusual thread spanning over a year about various Indian movies that have terrorist characters. A sample thread about terrorist attacks in Madrid is presented in Table 3.

The next query considered in 'Maran', an Indian politician who at one time was the Commerce Minister of India. This query produced various threads about his attending the WTO summit, various issues about Commerce discussed in

Date	Title
03/05/2001	Blast damages BBC office
03/06/2001	Britain placed on high alert
05/01/2001	McGuinness may admit role in IRA
06/02/2001	IRA arms dumps still intact, say inspectors
08/01/2001	N. Ireland: Loyalists threaten more violence
08/05/2001	Blast puts Ulster deal in doubt
08/08/2001	A breakthrough in Ulster
08/18/2001	Colombian links hit peace process
08/25/2001	Remove Sinn Fein from Executive, say hardliners

Table 1: Thread indicating IRA attacks.

Date	Title
04/07/2000	Elian' father arrives in U.S.
04/09/2000	Political pawn
04/11/2000	U.S. weighing options on Elian
04/16/2000	Court asked to order Elian handover
04/23/2000	U.S. enforces Elian-father reunion
04/24/2000	Widespread ire over re-union
04/27/2000	Senate panel may open hearings on Elian
05/01/2000	Elian case leaves impact on Florida politics
05/04/2000	Republicans not to press for hearings on Elian case?

Table 2: Thread corresponding to the Elian Gonzalez drama.

the Indian Parliament, etc. Table 4 shows an interesting thread corresponding to his health problems; note that this thread includes articles about his hospitalization and recovery in 2000, surgery in 2002, and his final hospitalization and demise in 2003.

Finally, we present two interesting threads from the query 'Nepal', which produced several interesting threads involving Indo–Nepalese relations, the politics and policy issues in the South Asian region. The first, presented in Table 5, spans six months, and is about the cancellation of Indian Airlines' flights to Katmandu, Nepal's capital, owing to security concerns, and the subsequent restoration of flights. The second, presented in Table 6, is a short thread that nevertheless spans nearly a year, and discusses silk smuggled through Nepal from China to India.

Date	Title
03/12/2004	190 killed in Madrid serial blasts
03/12/2004	'It looked like a platform of death'
03/12/2004	190 killed in Madrid serial blasts
03/13/2004	A challenge no Government can take lightly
03/14/2004	Spain follows up two leads
03/15/2004	Poll overshadowed by Al-Qaeda claim
03/16/2004	MEANING OF THE SPANISH VERDICT
03/17/2004	Power balance blown apart
03/17/2004	Blair 'vulnerable' after Aznar's defeat
03/21/2004	Spain: a vote to right a wrong
03/23/2004	Post-Cold War tasks
03/27/2004	A crumbling defence
03/27/2004	Old wine in old bottles

Table 3: Thread corresponding to Madrid attacks.

Date	Title
11/08/2000	'Maran needs pacemaker'
11/09/2000	Maran undergoes 'radical procedure'
11/10/2000	U.K. specialist examines Maran
11/11/2000	Maran 'on the road to recovery'
11/25/2000	Maran shifted out of ICU
12/01/2000	The 'desperate attempt' paid off
09/26/2002	Maran operated upon
06/16/2003	'Every possible step taken'
06/16/2003	Karunanidhi demands probe into medical treatment for Maran
09/08/2003	Karunanidhi calls on Maran
11/24/2003	Murasoli Maran dead
11/25/2003	Maran cremated with state honours
11/25/2003	Thousands pay homage to 'Murasoli' Maran
11/25/2003	'Nation has lost an eminent political personality'
12/03/2003	Nominate replacement for Maran: Vajpayee

Table 4: Thread corresponding to Mr. Maran, an Indian politician who was hospitalized but later recovered in 2002.

Date	Title
01/13/2000	'No IA flight to Kathmandu without fool-proof security'
03/07/2000	Team to review airport security in Nepal
04/04/2000	Talks on resuming flights inconclusive
04/25/2000	India for early decision on flights to Nepal
05/09/2000	Flights to Nepal may resume soon
05/11/2000	Resuming flights to Nepal
05/16/2000	IA to resume flights to Nepal from June 1
06/01/2000	IA flights to Kathmandu
06/02/2000	IA resumes service to Nepal

Table 5: Thread corresponding to IA flights to Nepal.

5. CONCLUSIONS AND FUTURE WORK

We presented a formal treatment of the problem of detecting threads in a time-stamped document collection. The formulation is based on maximizing the covering of a directed graph (at least b nodes) by a small number of (at most a) directed paths. We presented three algorithms for this prob-

Date	Title
03/03/2002	Chinese silk flooding local market
06/14/2002	Centre urged to end illegal import of Chinese silk
06/16/2002	Prevent dumping of Chinese silk, CM urges Centre
12/22/2002	Centre's decision a boost to sericulture sector
01/07/2003	'Imported silk ruining ryots'
01/08/2003	Centre urged to increase assistance for sericulture

Table 6: Thread corresponding silk smuggled through Nepal.

lem. The first algorithm solves the problem exactly for the case a is fixed and b is maximized, and is based on minimum cost flow. This algorithm, combined with a natural binary search, yields an exact solution to the problem of fixing b while minimizing the a parameter. The algorithm based on dynamic programming yields an exact solution to the case where $b = n$ and a is minimized. The algorithm based on bipartite maximum matching yields an exact solution to the case where b is maximized, with no constraint on the a parameter. We tested these algorithms on a large collection of news articles beginning January 01, 2000 to March 31, 2004. Our experiments showed that these algorithms are extremely effective in locating threads in these news articles. We also argued that the problem of thread identification goes beyond the realm of news articles—it can be applied to various temporal collections including patents, medical databases, legal documents, archives, scientific citations, etc. We believe that the results can be improved even more if one were to apply sophisticated term extraction methods that are suited to particular application domain.

It will be interesting to see if there is a purely combinatorial algorithm for the (a, b)-threads problem and if the problem can be solved (perhaps approximately) in the stream–sort model. Further work includes more general formulation of the problem, for instance, allowing small width trees (instead of just directed paths) so as to identify subthreads in a thread, or requiring that there are many edges between the initial and later portion of the thread so as to prevent topic drift. These problems fall under the purview of partition problems in directed graphs; for instance, see [20]. It will also be interesting to see the results of applying our algorithm to patent, medical, and legal articles.

6. REFERENCES

[1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proc. of the 45th IEEE Annual Foundations of Computer Science*, pages 540–549, 2004.

[2] R. Agrawal, S. Rajagopalan, R. Srikant, and Y. Xu. Mining newsgroups using networks arising from social behavior. In *Proc. of the 12th International Conference on World Wide Web*, pages 529–535, 2003.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.

[4] J. Allan. *Automatic Hypertext Construction*. PhD thesis, Cornell University, 1995.

[5] J. Allan, J. Carbonell, G. Doddington, J. Yamron, and Y. Yang. Topic detection and tracking pilot study: Final report. Proc. DARPA Broadcast News Transcription and Understanding Workshop, 1998.

[6] J. Allan, R. Gupta, and V. Khandelwal. Temporal summaries of new topics. In *Proc. of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 10–18, 2001.

[7] W. Blustein. *Hypertext Versions of Journal Articles: Computer-aided Linking and Realistic Human-based Evaluation*. PhD thesis, University of Western Ontario, 1999.

[8] T. Dalamagas. NHS: A tool for the automatic construction of news hypertext, 1998.

[9] T. Dalamagas and M. D. Dunlop. Automatic construction of news hypertext. In *HIM*, pages 265–278, 1997.

[10] J. Edmonds and R. M. Karp. Theoretical improvements in algorithm efficiency for network flow problems. *Journal of the ACM*, 19:248–264, 1972.

[11] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of web communities. In *Proc. 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 150–160, 2000.

[12] M. Garey and D. Johnson. *Computers and Intractability*. Freeman, 1979.

[13] A. Goldberg and R. Tarjan. Solving minimum-cost flow problems by successive approximation. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 7–18, 1987.

[14] S. J. Green. Automatically generating hypertext in newspaper articles by computing semantic relatedness. In *Proc. of the Joint Conference on New Methods in Language Processing and Computational Natural Language Learning*, pages 101–110, 1998.

[15] M. Henzinger, B.-W. Chang, B. Milch, and S. Brin. Query-free news search. In *Proc. of the 12th International Conference on World Wide Web*, pages 1–10, 2003.

[16] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 24(2):225–231, 1973.

[17] J. Kleinberg. Bursty and hierarchical structure in streams. In *Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 91–101, 2002.

[18] R. Kumar, U. Mahadevan, and D. Sivakumar. A graph-theoretic approach to extract storylines from search results. In *Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 216–225, 2004.

[19] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1481–1493, 1999.

[20] J.-J. Pan. *Path Partition and Its Variation in Graphs*. PhD thesis, National Chiao Tung University, 2004.

[21] A. F. Smeaton and P. J. Morrissey. Experiments on the automatic construction of hypertexts from texts. *The New Review of Hypermedia and Multimedia*, 1:23–39, 1995.

[22] D. A. Smith. Detecting events with date and place information in unstructured text. In *Proc. of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 191–196, 2002.

[23] R. Swan and J. Allan. Automatic generation of overview timelines. In *Proc. of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 49–56, 2000.

[24] E. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5:247–256, 1985.

[25] N. Uramoto and K. Takeda. A method for relating

multiple newspaper articles by using graphs, and its application to webcasting. In *Proc. of the 36th Conference on Association for Computational Linguistics*, pages 1307–1313, 1998.

[26] Y. Yang, T. Pierce, and J. Carbonell. A study on retrospective and on-line event detection. In *Proc. of the 21st ACM International Conference on Research and Development in Information Retrieval*, pages 28–36, 1998.