Generating Reading Orders over Document Collections

Georgia Koutrika ¹, Lei Liu ², Steve Simske ³

HP Labs, Palo Alto, USA

¹koutrika@hp.com ²lei.liu@hp.com ³simske@hp.com

Abstract—Given a document collection, existing systems allow users to browse the collection or perform searches that return lists of documents ranked based on their relevance to the user query. While these approaches work fine when a user is trying to locate specific documents, they are insufficient when users need to access the pertinent documents in some logical order, for example for learning or editorial purposes. We present a system that automatically organizes a collection of documents in a tree from general to more specific documents, and allows a user to choose a reading sequence over the documents. This a novel way to content consumption that departs from the typical ranked lists of documents based on their relevance to a user query and from static navigational interfaces. We present a set of algorithms that solve the problem and we evaluate their performance as well as the reading trees generated.

I. INTRODUCTION

There are two primary ways to access the contents of a document collection. A navigational interface provides an organized view of the contents. A search interface allows one to ask a query and see a dynamically generated ranked list of documents. Both these approaches may work fine when a user is trying to locate specific documents (e.g., ‘the homepage of University X?’) but they are insufficient when users need to access the pertinent documents in some logical order, for example for learning, research or editorial purposes.

Search engines hide document relationships while navigational interfaces capture only fixed relationships that do not dynamically adapt to a user’s specific need. In both cases, to determine which resources to read and in what order, users need to manually sift through the documents returned by the system. This can be a tedious process: an individual may read a significant number of documents, in the wrong order, until it is understood how they relate to each other, and then possibly re-read them in the right order to fully grasp their contents.

In this paper, we propose organizing documents in a reading order from general to more specific documents. This type of reading order is extremely useful in several contexts and applications. For instance, it can benefit researchers looking for papers on a particular topic, editors selecting articles to publish on a web site, IT personnel trying to figure out which applications. For instance, it can benefit researchers looking for papers on a particular topic, editors selecting articles to publish on a web site, IT personnel trying to figure out which applications.

We propose automatically determining the reading order of a set of documents from general to more specific documents based on their underlying relationships. We define two types of specificity relationships between two documents $d_1$ and $d_2$. If they cover the same topics at the same level, they are considered equivalent. If they are about related topics but $d_2$ is more focused on particular topics than $d_1$, then $d_2$ is more specific than $d_1$. Equivalent documents can be read in any order (or a reader can choose to read any of them). When a document is more specific than another document, then the former should follow the latter in reading order. For example, a document about classification methods should follow a document that is an introduction to data mining.

We quantitatively define specificity relationships with the help of two metrics: document generality and document overlap. We measure document overlap and generality based on the documents’ topical relationships. In other words, document specificity relations are determined by the document topical relationships. Our approach is domain-independent as it requires no external knowledge about the documents. We propose using the entropy as a measure of document generality and the weighted Jaccard for document overlap. Note that our reading order framework is independent of the metrics used.

We organize a set of documents in a reading tree that captures document specificity relationships. We define the reading order problem as the problem of computing a complete reading tree over a set of documents. Intuitively, a complete reading tree is a hierarchical tree where each node corresponds to one or more documents that are more general than the documents in any of their child nodes, and the edges capture the sequencing patterns between documents in the respective nodes. A reading tree follows a table-of-contents paradigm and hence it is easily understood by people.

Our approach to solve the reading order problem comprises of a topic model for modeling the topic relationships between documents, a topic calibration method for refining the results of the topic model and a tree generation algorithm that, as we prove, builds complete reading trees. The topic calibration method makes the whole approach more robust by correcting errors generated by the topic model. This method leverages the content similarity of documents to propagate document-topic scores between strongly similar documents on the account that due to their similarity they are more likely to share topics.

Evaluating the reading trees is non-trivial. Since this a novel problem, there is no ground truth. Furthermore, we need to find a suitable performance metric. For this purpose, we propose a new metric for comparing reading trees whose objective is to measure the relative order of two documents and
Contributions. The contributions of our work are as follows:

- We introduce the concept of ordering documents based on specificity relationships, which we define based on two topical measures, the document generality and topic overlap. We propose using the entropy for measuring the generality score of a document among other documents.
- We define the concept of a complete reading tree and we formally define the problem of determining the reading order of a set of documents.
- We present a reading tree construction algorithm that leverages the document topic relationships for building complete reading trees over a set of documents.
- To compensate for possible errors of the topic model, we propose a topic model calibration method that is designed to minimize an objective function similar to the one used for semi-supervised learning with local and global consistency.
- We evaluate the performance of the algorithms and the effect of the various parameters on the form of the reading trees and execution times.
- We introduce an evaluation metric that is suitable for comparing reading trees and we apply different strategies for evaluating the generated reading trees.

II. RELATED WORK

A document collection can be accessed through navigational interfaces and search engines. Navigational interfaces provide a fixed view over a known collection. Existing search engines, such as Google, and Bing, rank documents based on their relevance to a query. These ranked lists do not suggest any reading order among the documents. Google’s advanced search interface1 organizes search results into three reading levels: basic, intermediate, and advanced. However, the reading level classification provides only a very coarse document ordering.

Clustering-based search engines, such as Carrot2, cluster search results allowing the users to elaborate their initial query by focusing on a specific cluster. There is substantial prior work on hierarchical document clustering methods, including techniques for iteratively partitioning (merging) the dissimilar (similar) documents [1], incremental clustering [2], and probabilistic taxonomy modeling [3]. Hierarchical clustering trees are inherently different from our reading trees: any level of a clustering tree is a segmentation of all documents whereas any level of a reading tree maps a unique subset of the input documents. Hence, in a clustering tree, a document appears at all levels of the hierarchy. On the other hand, a reading tree defines a partial order on the input documents.

Instead of clustering the documents into groups that are assigned to the nodes of a tree, label tree learning methods aim at classifying documents in multiple classes [4], [5], [6]. Classes are automatically organized into a label tree that captures the relationships between them. A label tree generates a segmentation of the training documents but it does not specify any order among these documents.

Efforts to understand a corpus of documents include corpus summarization approaches [7], [8]. Ordering documents in some meaningful way has been studied in the context of news articles [9], [10]. The method described in [10] starts with two news articles and automatically finds a topic-based coherent chain of articles linking them together. Incident threading is the process of identifying incidents in a news stream and connecting them through contextual links, such as ‘consequence of’ or ‘follow-up’ [9]. These approaches work with particular types of documents (e.g., articles that describe strong events [9]). Our approach is general-purpose and domain-independent, and it aims at ordering a corpus of documents based on the document specificity relationships, which is a novel approach to corpus analysis.

A related line of research is the generation of personalized curriculum sequences in e-learning systems [11], [12], [13], [14]. Curriculum sequencing can be seen as a two-part process: deciding on relevant topics based on the current student model, which captures the student’s goals and performance, and then selecting the best course module(s) to show to the student [15]. The main shortcoming of these methods is that they are designed either for a specific course [13], [14] or when the domain knowledge is available [11], [12]. Therefore, they can not be extended to ordering random document sets.

The concept of ‘specialization/generality’ is inherent in navigation hierarchies (e.g., IS-A hierarchies in ontologies), where the direction of the edges usually implies that an entry closer to the root is more general than a descendant deeper in the hierarchy. However, quantifying how general one entry is relative to another one is not trivial. Most existing techniques extract these relationships either (a) from the information content of the terms in an ontology (computed over a large corpus) [16] or (b) from the structure (e.g., density, depth) in the hierarchy itself [17]. A third way leverages the keywords contained in the entries to decide the degree of generality based on the relative content of two documents [18], [19].

We also rely on the textual content of documents but we model documents using topics. We do not assume a pre-existing hierarchy. Our objective is to automatically organize

---

1https://www.google.com/advanced_search (last accessed October 2014)
2http://search.carrot2.org/
a collection of documents in a reading tree where each node corresponds to one or more similar documents and the edges capture general-to-more-specific sequencing patterns between documents. To the best of our knowledge, there is no prior work for generating such reading sequences over documents.

III. SPECIFICITY RELATIONSHIPS BETWEEN DOCUMENTS

There are different types of document relationship that can determine the order in which two documents should be read. For example, two documents on the same topic may be ordered based on their importance: the most important document should be read first. Another example is a prerequisite relationship: a document on probabilities should be read before a document on Bayesian networks. In this paper, we focus on the specificity relationships between documents, which are determined based on their contents.

We identify two types of specificity relationship, sequencing and equivalence. We first give an intuitive definition. Later we define them more formally.

A specificity equivalence relation \( d_i \leftrightarrow d_j \) signifies that \( d_i \) and \( d_j \) cover the same topics at the same level. We say that \( d_i \) and \( d_j \) are equivalent based on specificity.

A specificity sequencing relation \( d_i \rightarrow d_j \) signifies that \( d_i \) and \( d_j \) have some overlap but \( d_j \) is more specific or focused than \( d_i \). We say that \( d_i \) precedes \( d_j \) based on specificity.

As an example, consider the following documents: \( d_1 \) is an introduction to data mining, \( d_2 \) describes classification methods, and \( d_3 \) is another introductory paper on data mining. Both \( d_1 \) and \( d_3 \) cover the same topic to a similar extent and hence they have a specificity equivalence relation, \( d_1 \leftrightarrow d_3 \). Consequently, one can choose to read any of them. However, \( d_2 \) is more focused, i.e., it has a specificity sequencing relation to the other documents: \( d_1 \rightarrow d_2 \) and \( d_3 \rightarrow d_2 \).

We define two specificity metrics:

- The generality score of a document is a function \( g: D \rightarrow \mathbb{R} \) that computes the generality of a document. It holds that \( d_i \) is more general than \( d_j \), if \( g(d_i) \geq g(d_j) \).
- The overlap score of a pair of documents is a function \( o: D \times D \rightarrow [0, 1] \) that computes the degree of their commonality. Score equal to 0 means no overlap, while 1 means maximum overlap.

We now formally define specificity relations as follows:

\[
\text{Equivalence: } d_i \leftrightarrow d_j \text{ if } |g(d_i) - g(d_j)| \leq \kappa \land o(d_i, d_j) \geq \tau
\]

\[
\text{Sequencing: } d_i \rightarrow d_j \text{ if } g(d_i) > g(d_j) \land o(d_i, d_j) > 0 \land ((g(d_i) - g(d_j)) > \kappa \lor o(d_i, d_j) < \tau)
\]

\( \tau \) defines the minimum overlap between two equivalent documents and \( \kappa \) defines the maximum difference of their generality scores.

A. Measures of document generality and overlap

We measure document overlap and generality based on the documents’ topical relationships. Topic models [20] represent documents as mixtures of topics, where a topic is a probability distribution over words. A topic model aims at discovering the hidden thematic structure of a collection of documents by finding how topics are assigned to documents, and how topics are described by words in the documents. Representing a document using topics rather than document keywords allows capturing implicit relationships between documents, not just the explicit similarity of their common words.

Given a collection \( D \) of \( n \) documents, a topic model generates \( s \) topics, \( t_1, \ldots, t_s \), that describe this collection. \( F_{n \times s} \) is the document-topic matrix that captures how the \( s \) topics are assigned to the \( n \) documents. We denote \( F_t \) the topic vector associated with \( d_i \). \( F_{im} \in [0,1] \), with \( i \leq n \) and \( m \leq s \), is the probability that topic \( t_m \) is assigned to document \( d_i \).

Based on the above, the generality score is defined as a function over \( F_{n \times s} \), i.e., \( g: F_{n \times s} \rightarrow \mathbb{R} \). Similarly, the overlap score of a pair of documents is defined as \( o: F_{n \times s} \times F_{n \times s} \rightarrow [0,1] \).

Document generality. We can measure the document generality based on the document’s entropy over the topics. The basic intuition behind the entropy is that the higher a document’s entropy is, the more topics the document covers hence the more general it is. Using the Shannon entropy, the generality score \( g(d_i) \) of document \( d_i \) is computed as follows:

\[
g(d_i) = H(d_i) = \sum_m -F_{im} \log(F_{im})
\]

Document overlap. The overlap of two documents can be computed as their weighted Jaccard score [21]. The weighted Jaccard extends the classic Jaccard index, which is defined as the size of the intersection divided by the size of the union of the topic sets assigned to each document, by taking into account their topic probabilities. The overlap score can be computed as follows:

\[
o(d_i, d_j) = \text{Jaccard}(d_i, d_j) = \frac{F_i \cdot F_j}{|F_i|^2 + |F_j|^2 - F_i \cdot F_j}
\]

Higher overlap scores indicate more common topics between the documents.

Example 1. To illustrate the document generality and overlap using the entropy and Jaccard score, respectively, we consider 6 Wikipedia documents related to “Machine Learning” as shown in Table I, and we use the content of the corresponding Wikipedia page as input to the topic model. Table II shows the assignment of five topics to the six documents. To get an idea of the meaning for each topic, we show the top five terms for each topic in Table III. Consider, for example, documents \( d_1 \) and \( d_3 \). \( d_1 \) covers all topics almost to the same extent whereas \( d_3 \) focuses on topic2. Their overlap is \( o(d_1, d_3) = 0.33605 \) and their generality scores are \( g(d_1) = 0.6494 > g(d_3) = 0.53066 \). Hence, \( d_1 \) (“Machine Learning”) is more general than \( d_3 \) (“Support Vector Machine”) which is actually true. As another case, documents \( d_2 \) and \( d_5 \) have very small overlap, \( o(d_2, d_5) = 0.2289 \). On the other hand, their generality scores are close, \( g(d_2) = 0.53748 \) and \( g(d_5) = 0.60335 \). This indicates that \( d_2 \) (“Supervised Learning”) and \( d_5 \) (“Unsupervised Learning”) will end up at
different branches of the reading tree but probably at the same level, which reflects their actual relation. 

Note that other metrics for measuring document generality and overlap are possible. For example, instead of the Shannon entropy, we could use the residual entropy (entropy of non-common terms). Given a set of estimated topic probabilities for each document, the overlap of two documents can be compared using similarity measures such as the cosine similarity or the unnormalized dot product. The algorithms we present are independent of how generality and overlap are measured.

IV. THE READING ORDER PROBLEM

A reading graph $R(V, E)$ over a document set $D$ is a directed acyclic graph whose nodes correspond to the input documents and edges capture specificity relations among the documents. In particular, a node $v_i \in V$ maps a non-empty set $D_i \subseteq D$ of equivalent documents. An edge $v_i \rightarrow v_j$ between nodes $v_i$ and $v_j$ signifies that documents belonging to the corresponding document set $D_i$ precede documents belonging to the respective set $D_j$.

A complete reading tree over a document set $D$ is a reading graph $R(V, E)$ with the following properties:

(a) For each node $v_i \in V$ with $D_i$, being its corresponding set of documents, it holds that: a document $d \in D$ maps to $v_i$ iff $d \leftrightarrow d_i$, for all $d_i \in D_i$.

(b) For each pair of nodes $v_i, v_j \in V$ with $D_i$ and $D_j$ being their sets of documents, and an edge $v_i \rightarrow v_j$, it holds that: For each pair of documents $d_i \in D_i$ and $d_j \in D_j$, it is $d_i \rightarrow d_j$.

(c) For each pair of nodes $v_i, v_j \in V$ with $D_i$ and $D_j$ being their sets of documents, and an edge $v_i \rightarrow v_j$, it holds that: there is no other node $v_k$, such that: $v_k \rightarrow v_j$.

A reading sequence $d_{m_1} \rightarrow d_{m_2} \cdots \rightarrow d_{m_k}$ of documents $d_{m_i} \in D_i$, $i = 1 \ldots k$, maps to a path $v_{l_1} \rightarrow v_{l_2} \cdots \rightarrow v_{l_k}$ on the graph with $v_{l_i} \in V, i = 1 \ldots k$ such that $d_{m_i} \in D_{l_i}$ of node $v_{l_i}$, $i = 1 \ldots k$.

Figure 1(a) shows an example reading tree over a set of six documents. Several reading sequences may be derived over the document relationships. The figure shows an example reading sequence: $d_1 \rightarrow d_4 \rightarrow d_6$. Furthermore, there may be more than one way to represent the same set of documents as a complete reading graph. For example, consider documents $d_1$, $d_2$ and $d_3$, which have some overlap and $d_1 \leftrightarrow d_2$ and $d_2 \leftrightarrow d_3$ but $d_1 \not\leftrightarrow d_3$. Figure 1(b) shows two possible complete reading trees. Finally, multiple reading trees may be needed to cover a document collection. For instance, documents with no overlap will be mapped to different reading trees.

The reading order problem is defined as follows:
A. Topic Model

Topic modeling algorithms are statistical methods that analyze the words of the original texts to discover the themes that run through them, how those themes are connected to each other, and how they change over time [20], [22], [23], [24]. The idea behind a topic model is that when a document is about a particular topic, some words should appear more frequently. Hence, documents are mixtures of topics, where each topic is a probability distribution over words. Each topic is about a particular topic, some words should appear more frequently. Hence, documents are mixtures of topics, where each topic is a probability distribution over words.

The topic model algorithm generates a document-topic matrix \( F \in \mathbb{R}^{n \times s} \), where each matrix element captures the document-topic membership as a probability. In this paper, we use Latent Dirichlet Allocation (LDA) with Gibbs sampling [22]. In this model, the number \( s \) of topics to be generated is given as input to the algorithm and it depends on the document set. A small number of topics could provide a broad overview of the document structure whereas a large number could provide fine-grained topics at the cost of computational time. We treat the number of topics as one of the parameters of our algorithms and analyze its impact in the experiments.

B. Topic Model Calibration

Despite their powerfulness, topic models have a number of known issues. Among those, two are important for our problem setting: when same documents end up having not exactly the same topics [25], and when topics may be missed or misassigned [26]. The topic model calibration aims at ameliorating these issues by using the explicit textual similarities of documents to influence the topic assignment. This method allows the topics of a document to be influenced by the topics of its most similar neighbors. We first compute the document similarity graph that captures the explicit similarities of the documents based on their keywords. Then we present a topic score propagation method that smooths the initial topic distribution of a document based on those of its neighbors on the similarity graph.

1) Similarity Graph Construction: Given the document-term matrix \( X \in \mathbb{R}^{n \times m} \) for the document set \( D \), we can compute an \( n \times n \) kernel matrix \( K \) for all pairs of documents in \( X \), where each entry \( K_{ij} \) is a numeric value reflecting the similarity between two documents \( d_i \) and \( d_j \). Any kernel method can be used for this purpose. Without loss of generality, we use the cosine similarity as \( K(x_i, x_j) = \frac{x_i \cdot x_j}{\|x_i\| \|x_j\|} \).

The document similarity graph \( G = (V', E', W) \) for the set \( D \) is an undirected weighted graph where \( V' \) is the set of nodes, each node mapping to a document \( d_i \in D \), \( E' \) is the set of the edges, and \( W \) is an \( n \times n \) adjacency matrix. For each pair of documents \( d_i \) and \( d_j \), there is an edge \( e_{ij} \in E' \) connecting the respective nodes with weight \( W_{ij} = K(x_i, x_j) \) if and only if \( K(x_i, x_j) \geq \epsilon \). \( \epsilon \) is a parameter to control the sparseness of the graph. The higher the value of \( \epsilon \), the sparser the graph is as fewer links are present that correspond to the more similar documents. In this paper, we assign \( \epsilon = 0.3 \). For simplicity hereafter, we will use the term document similarity graph \( G = (V', E', W) \) implying that it is an \( \epsilon \)-neighborhood document similarity graph.

Apart from content similarity, links between documents, such as web links and citations, also indicate that the documents are related to some extent. When such links exist, we could build a document similarity graph by combining both content and link information [29].

Algorithm 1 Score Propagation

Input: \( G = (V', E', W) \), \( F^{(0)} \), \( \beta \) and MaxIter
Output: \( F \): topic probabilistic scores of nodes in \( G' \)
Build \( W \) using kernel matrix \( K(x, x) \)
Calculate \( \hat{W} = D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \)
for \( i = 1 \) to \( \text{MaxIter} \)
    Updating topic scores \( F \) using equation (7) as:
    \[
    F = \beta \hat{W} F + (1 - \beta) F^{(0)}
    \]
end for

2) Topic Score Propagation: Both the document-topic matrix and the similarity graph are input to the score propagation module, whose target is to propagate the document-topic assignments over the graph. Score propagation is inspired from label propagation that leverages the idea that strongly connected nodes should share similar class information to label nodes in a graph [27], [30]. In our setting, strongly similar documents have a high chance to share similar topics, and our target is to learn the topics in the similarity graph rather than perform network node classification task.

The basic idea of score propagation is the following: after getting the initial topic distribution across the documents using the topic model algorithm, we propagate the topic probabilistic scores over the document similarity graph \( G = (V', E', W) \), so the potential topics of a document take into consideration the topic probabilistic scores of their neighbors (which in turn, depend on the scores of their respective neighbors, and so on). The algorithm iteratively updates the topic probabilistic scores.
of a node in the document-topic matrix $F_{n \times s}$ based on the weighted average of the scores of its neighbors.

Given the document-topic matrix $F_{n \times s}$ and the document similarity graph $G = (V', E', W')$, our score propagation algorithm is formally designed to minimize an objective function similar to the one used for semi-supervised learning with local and global consistency [27]. Our objective function is:

$$Q(F) = \frac{1}{2} \sum_{ij} W_{ij} \left[ \frac{F_i}{\sqrt{D_{ii}}} - \frac{F_j}{\sqrt{D_{jj}}} \right]^2 + \frac{\mu}{2} \sum_i (F_i - F_i^{(0)})^2$$

where $F_i$ is a vector that maps to the $i^{th}$ row of the matrix $F_{n \times s}$ and captures the topic distribution for document $d_i$, and $F_i^{(0)}$ corresponds to the initial topic probabilistic scores determined by the topic model. $W_{ij}$ is the similarity weight for two documents $d_i$ and $d_j$. Finally, $D$ is a diagonal matrix whose diagonal elements are given by $D_{ii} = \sum_j W_{ij}$.

Intuitively, the first term in the objective function ensures that the topic probabilistic scores for any pair of nodes connected by a highly weighted link should not differ substantially. The second term ensures that the scores of the nodes should not deviate significantly from their initial values. The parameter $\mu$ controls the tradeoff between the two terms.

Let us now see how we can solve the objective function and arrive to the updating formula for the matrix $F$. First, our objective function can be rewritten in the following form:

$$Q(F) = \frac{1}{2} \sum_{ij} W_{ij} \left[ \frac{F_i^2}{D_{ii}} - 2 \frac{F_i F_j}{\sqrt{D_{ii} \sqrt{D_{jj}}}} + \frac{F_j^2}{D_{jj}} \right] + \frac{\mu}{2} \sum_i (F_i - F_i^{(0)})^2$$

$$= \frac{1}{2} \sum_i \left( F_i^2 - 2 \sum_j W_{ij} F_j \frac{F_i}{\sqrt{D_{ii} \sqrt{D_{jj}}}} + \sum_j F_j^2 \right) + \frac{\mu}{2} \sum_i (F_i - F_i^{(0)})^2$$

Now, we can express the objective function in matrix notation as follows:

$$Q(F) = F^T(I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) F + \frac{\mu}{2} \| F - F^{(0)} \|^2$$

$$= F^T \hat{L} F + \frac{\mu}{2} \| F - F^{(0)} \|^2$$

where $\hat{L} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}}$ is the normalized Laplacian of the graph.

To optimize the objective function, we take its partial derivative with respect to $F$ and set it to zero:

$$\frac{\partial Q(F)}{\partial F} = (I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}) F + \mu (F - F^{(0)})$$

$$= (1 + \mu) F - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} F - \mu F^{(0)} = 0$$

The preceding equation can be re-written in the form of an iterative update formula:

$$F = \frac{1}{1 + \mu} \hat{W} F + \frac{\mu}{1 + \mu} F^{(0)} = \beta \hat{W} F + (1 - \beta) F^{(0)}$$

where $\hat{W} = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ is the normalized adjacency matrix and $\beta = \frac{1}{1 + \mu}$ is a damping factor that controls the tradeoff between biasing the scores according to the graph structure as opposed to the initial score matrix $F^{(0)}$. If $\beta = 0$, the topic scores are equal to the initial values obtained from topic model. On the other hand, if $\beta = 1$, the topic score of a node depends only on the scores of its neighbors.

Since $\beta$ determines the contributions of the neighbors of a node in a graph (while $1-\beta$ determines the contribution of the node itself), $\beta$ operates in a similar fashion as the damping factor used in the PageRank formula. For Web graphs, the parameter $\beta$ is often set to 0.85 [31]. We use the same value throughout our experiments.

The score propagation method is shown in Algorithm 1.

### C. Reading Tree Generation

The tree generation process progressively weaves the ordering for a set of documents by determining the specificity relations among the documents. It takes as input the set of documents, the document-topic assignments $F$, and two parameters: $\tau$, which defines the minimum overlap between two equivalent documents, and $\kappa$, which defines the maximum difference of their generality scores. Both parameters are used to define the specificity relations (Section III). We use the formulas (3) and (4) for computing document generality and overlap, respectively, but the algorithm is really independent of how document overlap and generality are estimated.

The algorithm builds a complete reading tree in an iterative way. At each round, it handles a subset of similar documents that need to be connected to the tree already created. The algorithm’s intention is to grow a sub-tree out of this set of documents and connect it to the existing tree. For this purpose, it first creates the root of this new sub-tree by putting together the most general documents from the set in consideration that are also equivalent. The remaining documents of this set are clustered such that they have some overlap with the root and among them. Each cluster becomes a new set of documents out of which the algorithm will further create new nodes and edges. This process repeats until no more tree growing is possible and there are no documents unprocessed. Initially, the set of documents under consideration contains all documents and the current root node is a dummy node. The algorithm is outlined in Algorithm 2.

The algorithm starts by computing the generality score for each input document and the generality difference matrix $E$ where $E_{ij} = |g(d_i) - g(d_j)|$ indicates the generality difference of documents $d_i$ and $d_j$. All documents are then ordered in descending order of their generality score. A dummy node is created and this node becomes the root of the tree. It is also the first node from where the tree will start to expand (called expansion node). The core operations of the algorithm are described as the function $DynOrder$, which is executed repeatedly but for a different subset of the input documents and growing the tree from a different expansion node each time. Its objective is to build a tree out of its input documents and connect it to the expansion node $v_r$. Steps 1-4 are responsible for the creation of the root node of this tree. This node, $v_s$, groups the more general documents from the current set of
documents that have the required topic overlap and generality closeness based on the equivalence condition (1).

To build this node, the process starts with the most general document \( d \). Subsequently, it considers documents in descending order of generality. As long as a document \( d_j \) has a close generality score to \( d \), and its overlap with all the documents already selected for the node \( v_s \) satisfies the equivalence criterion, this document \( d_j \) is also added to the set of documents for the node \( v_s \). The node \( v_s \) becomes the root node for the remaining documents in the current set and it is connected to the node \( v_r \) at step 5.

The remaining steps are responsible for re-organizing the rest of the documents in groups such that each group will be used to further grow the tree in a different direction. Step 7 creates a set \( C \) of all documents that have non-zero overlap with at least the most general document \( d \) of the node \( v_s \) just created. The reason is that any document in any node at any level under \( v_s \) should have even a distant relation to the documents of \( v_s \). Step 8 divides \( C \) into sets \( D_e \), such that the documents contained in each set have non-zero overlap among them. The reason is that the nodes of a tree should have some relatedness. The node \( v_s \) becomes the new expansion node. The function \( \text{DynOrder} \) is called for each set \( D_e \) to grow a new reading tree. Each of these trees is connected to \( v_s \).

For simplicity in presentation, Algorithm 2 shows the case of constructing one reading tree. The algorithm builds more than one tree if required. The critical point is the output of step 7. If this is an empty set but there are still documents whose reading order has not been determined, then the algorithm starts a new set of rounds building a new tree that is connected to the dummy root node for the remaining documents.

**Theorem.** The algorithm builds complete reading trees.

**Proof.** We first show that steps 1-4 create a node of equivalent documents. Each pair of documents in the node satisfies the topic overlap condition. Furthermore, each document \( d_j \) satisfies the generality condition wrt the most general document \( d \), i.e., \( g(d) - g(d_j) < \kappa \). This condition is sufficient for making sure that all pairs of documents in the node satisfy the generality condition since they are selected in descending order of their generality score. It is obvious that the first time these steps are executed, a node containing all documents equivalent to \( d \) is created. We now show that each subsequent iteration of the steps also groups all equivalent nodes together satisfying property (a) of a complete reading tree.

Let \( S \) be the set of documents of a node and \( d \) be its most general document. Let us assume that there is a document \( d' \) such that \( g(d) - g(d') < \kappa \) and \( o(d_i, d') > \tau \forall d_i \in S \). However, step 8 groups together all documents that have non-zero overlap \( o(d_i, d_j) > 0 \). For each such group \( D_e \), steps 1-4 place all documents that satisfy the equivalence condition to the same node. Hence, \( d' \) would have been included in \( S \).

Consequently, the algorithm builds all nodes of equivalent documents. We now need to show that it also builds all sequencing relations satisfying properties (b) and (c) of the complete reading tree definition. Step 5 connects a newly created node \( v_s \) to the current expansion node \( v_r \). The first time the algorithm runs, \( v_r \) is the dummy root. In subsequent rounds, \( v_r \) has been created earlier by steps 1-4. Let \( R \) be the documents for \( v_r \), and \( S \) for \( v_s \). Since the algorithm examines documents in decreasing order of generality, it holds that \( g(d) > g(d') \) for any pair of documents \( d, d' \in R \). If \( g(d) - g(d') > \kappa \), then it is easily shown that the sequencing relation between \( d \) and \( d' \) holds. Let us assume that \( g(d) - g(d') < \kappa \). Then, \( d' \) could not have had enough overlap with the documents in \( R \), otherwise it would have been grouped there. Hence, for the nodes \( v_r, v_s \), their corresponding documents are in a sequencing relation. Hence property (b) of the complete reading graph holds.

Property (c) is easily shown. Let us take the simple case of \( d_i \rightarrow d_j \). If there was a document \( d_k \) such that \( d_i \rightarrow d_k \rightarrow d_j \), then it would be \( g(d_i) > g(d_k) > g(d_j) \). But then \( d_k \) would have been picked before \( d_j \).□

**Example 2.** Let us illustrate the tree generation algorithm with the help of Figure 3. Figure 3(a) shows the document-topic matrix. The document generality scores are as follows:

```
Algorithm 2 Reading Tree Generation
Input: doc set \( D \), doc-topic matrix \( F \),
      generality difference threshold \( \kappa \), overlap threshold \( \tau \)
Output: Tree Structure

foreach \( d_i \) in \( D \)
  Calculate generality score \( g(d_i) \)
end for

Calculate generality score difference matrix \( E \)
Create a dummy root node \( v_r \)
DynOrder(\( D, F, \kappa, \tau, v_r \))
```

**Function DynOrder**

Input: doc set \( D \), doc-topic matrix \( F \),
      generality score difference matrix \( E \),
      generality difference threshold \( \kappa \),
      overlap threshold \( \tau \), expansion node \( v_r \)
Output: Tree Structure

while \( D \neq \emptyset \) do
  1. Create a set \( S \) containing the most general \( d \) in \( D \)
  2. Select the next most general document \( d_j \) in \( D \)
  3. while \( o(d_i, d_j) > \tau \) and \( g(d) - g(d_j) < \kappa \)
     if \( o(d_i, d_j) > \tau \) \( \forall d_i \in S \) then add \( d_j \) to \( S \)
     Select the next most general document \( d_j \) in \( D \)
end while
  4. \( S \) contains the most general equivalent documents and it is mapped to a new node \( v_s \)
  5. Connect node \( v_s \) to the expansion node \( v_r \)
  6. Remove from \( D \) all documents belonging to \( S \)
  7. \( C \leftarrow \{ d_j \in D \mid o(d_i, d_j) > 0 \} \)
  8. Divide \( C \) into clusters \( D_e \) s.t.:
     for each \( D_e \) it holds \( o(d_i, d_j) > 0, \forall d_i, d_j \in D_e \)
  9. foreach cluster \( D_e \)
     DynOrder(\( D_e, F, \kappa, \tau, v_s \))
end for
end while
To summarize, the overall complexity of our approach is $O(n^2 \log n)$ and could be further reduced to $n \log n$.

VII. Experiments

A. Datasets and Setup

We use a variety of datasets. To understand the effect of the tree generation parameters and how our methods scale, we use DB, a set of 25000 research papers from subsequent, recent years from TKDE, VLDB, CIKM, and SIGMOD, and NEWS, a set of 20000 news articles from 20 newsgroups. The latter is the 20 Newsgroup benchmark, a popular dataset for experiments in text application of machine learning. For evaluating the output of our approach, we use DB, NEWS, and WIKIPEDIA. By using different datasets, we also see how well our method performs for different data and for different applications. All the experiments were performed on a single PC with Intel Core i7 CPU 2.6GHz and 8 GB memory, running Windows 7 operating system.

B. Tree Generation Parameters

The tree shape. Our reading trees are shaped by three parameters: (a) the number $s$ of topics used to describe the document set, (b) the minimum topic overlap $\tau$ between two equivalent documents, and (c) the maximum difference $\kappa$ of their generality scores. By controlling these parameters, we can grow different trees that capture document relations in a more or less fine-grained way. At the same time, these parameters affect execution times of the various algorithms. In this section, we present experiments on the impact of the parameters.

We have performed several experiments with different combinations of the parameters and for different subsets of the DB and NEWS datasets. Here, we discuss representative experiments with parameters set as follows: $s \in \{10, 20, 30, 40, 50\}$, $\tau \in \{0.5, 0.7, 0.9\}$, and $\kappa \in \{0.001, 0.005, 0.01\}$.

To see how the parameters affect the features of the tree, we measured the number of nodes created in a tree. Figure 4 reports the results on a VLDB sample of 400 papers spanning approximately 5 consecutive years (the trends are similar across different datasets). In Figure 4(a), we observe a smooth uplift in the number of the nodes created with increasing number of topics and the same $\kappa$. On the other hand, smaller $\kappa$ creates significantly larger number of nodes. This is intuitive because the generality score captures noise

\[ g(d_3) = g(d_4) = -4 \times 0.25 \times \log(0.25) = 0.602059, \]

\[ g(d_1) = g(d_2) = -2 \times 0.5 \times \log(0.5) = 0.30102, \]

\[ g(d_1) = -1 \times \log(1) = 0. \]

Let us use $\kappa = 0.2$, $\tau = 0.8$. Figure 3(b) shows the algorithm progress and its final output.

$d_3$ is selected as it has the largest entropy score. $d_4$ is also selected because $g(d_3) - g(d_4) = 0 < \kappa$ and $o(d_3, d_4) = 1 > \tau$. $d_3, d_4$ together become the root node of the tree. The remaining documents form two groups based on their overlap: $d_1$ and $d_5$ comprise one group, and $d_2$ comprises another because $o(d_1, d_5) = 0.5$, $o(d_1, d_2) = 0$ and $o(d_5, d_2) = 0$. The tree will grow two branches, one from each group, by repeating the same process. For the cluster of $d_1$ and $d_5$, we select $d_1$ as the most general document among the two. However, $g(d_1) - g(d_3) = 0.30102 > \kappa$, hence, the new node that will be added under the node $\{d_5, d_4\}$ contains only $d_1$. The algorithm continues iterating through these steps.

D. Complexity Analysis

We now analyze the time complexity of the methods. The topic model complexity is $O(n \times s \times \text{Iter})$ [32], where Iter is the number of iterations, $n$ is the number of documents and $s$ is the number topics to be generated. For large document sets, $s$ and Iter are relatively small constant values, thus the complexity of the topic model can be seen as $O(n)$.

The topic model calibration has two parts. The complexity of graph construction through pairwise similarity calculation is $O(n^2)$, where $n$ is number of documents. We could reduce the complexity of $\epsilon$-neighborhood graph construction algorithm from quadratic to linear, $O(n)$, with the method proposed in [33]. Regarding the score propagation component, the time complexity is $O(\text{MaxIter} \times n^2 \times s)$, where MaxIter is the maximum number of iterations. In this work, we found that our score propagation could converge for a small number of iterations (less than 20), so we set MaxIter = 20. Then, the overall complexity of score propagation for the dense graph is $O(n^2)$. However, in this paper, two documents are connected only if their similarity is larger than $\epsilon = 0.3$, which leads to a sparse graph (the matrix $W$ of Algorithm 1 is sparse). Consequently, the time complexity can be rewritten as $O(\text{MaxIter} \times n \times c \times s)$, where $c$ is the average degree of the nodes in the graph, which is a much smaller value than $n$.

The complexity of reading tree generation is $O(n^2 + d)$, where $d$ is the depth of the tree. In general, it is $d \approx \log n$ so the complexity for tree construction can be $O(n^2 \log n)$. We could reduce the time complexity as $O(n \log n)$ by using a more efficient algorithm for the clustering component.
Hence, we ignore score propagation in the results below. Basically equal to the similarity graph computation time. In all experiments, and the topic model calibration time is the execution times. The score propagation time is negligible.

Figure 4(b) is quite different. Interestingly, the number of nodes is not proportional with the number of topics. We create the largest number of nodes with an average number of topics and an average overlap threshold $\tau$. Too few topics provide a very coarse description for the documents making several ones looking similar. Too many topics makes hard to understand which documents are really similar and hence many documents may be placed in a single node. Hence, in both cases, we do a poor job recognizing document relationships.

From the two graphs of Figure 4, we observe that the stricter the generality constraint is the more nodes are created in the tree. On the other hand, the number of topics in combination with the overlap constraint can affect the number of tree nodes.

The number of nodes in the tree shows how a tree is shaped between two extremes: all documents being grouped in a single node versus each document being a different node. Another parameter that describes the shape of the tree is branching. Branching essentially shows how a tree is shaped between being very deep (e.g. a chain) or very shallow. In our experiments, we found that a better way to capture the tree shape, instead of using the branching factor of a tree, is by measuring the number of leaf nodes. The reason is that the branching factor may vary across a tree and an average value is not a good indicator. Our experiments on how the tree generation parameters affect the number of tree leaf nodes have showed that the number of topics and the overlap constraint are defining factors: stricter overlap or fewer topics lead to deeper trees. We do not show the corresponding graphs due to space considerations.

**Execution time.** We now examine the effect of parameters on the execution times. The score propagation time is negligible in all experiments, and the topic model calibration time is basically equal to the similarity graph computation time. Hence, we ignore score propagation in the results below. Figure 5 reports times for two datasets: the left column shows $VLDB$ (400 papers) and the right column shows $CIKM$ (150 papers). We observe that in all graphs the topic model time does not vary a lot compared to other components’ times, and it increases relatively slowly with the number of topics. The similarity graph computation always comprises an important part of the total execution time and depends on the number of documents: for $VLDB$, the average time is 100 seconds, while for $CIKM$ is 30 seconds.

From all the execution times, the tree generation time is the most important. The reason is that the topic model computation and the similarity graph computation can be performed offline for a document collection while the tree generation can be performed online for the desired subset of documents depending on the application. The tree generation is quite efficient. The generation time grows with the number of topics and the overlap $\tau$ but it shrinks with the generality parameter $\kappa$. Increasing the topics or $\tau$ means that more documents may be placed at the same node hence more pair-wise comparisons are needed to check the document overlap. On the other hand, shrinking $\kappa$ shrinks the number of documents per node and hence the number of pair-wise comparisons.

Our discussion above has focused on research papers. These are long documents. Figure 6(a) shows how the size of the document affects execution times based on two datasets of 500 documents each: $TKDE$ contains papers from the journal, and $NEWS$ contains short news articles. We observe that similarity graph times are affected by the length of the document: long papers contain many terms generating a big document-term matrix. Tree generation is also longer for the longer $TKDE$ documents because the tree created has 190 nodes almost twice the size of the $NEWS$ tree.

In the experiments above, we have considered the full document for each paper. In practice, the introduction and related work sections are sufficient to build the reading tree and the execution times are significantly improved. We performed experiments to compare the outputs in both scenarios: the generated topics and trees were equally good (in Section VII-D we explain how we compare different trees).

**C. Scalability**

We tested how our approach scales with the number of documents for the $DB$ and $NEWS$. Generally, we observed that the overall execution time is less than 1 minute for 1000 documents, a relatively large number: 50 seconds for $DB$ when processing the whole article, and 6 seconds for $NEWS$.

Figure 6(b) shows execution times for three sets of documents from the $NEWS$ dataset with size 1000, 10000 and 20000, respectively. We observe that as the number of documents increases, the topic model and the similarity graph computation are the most time-consuming components.

In practical scenarios, it does not make sense to compute a reading tree over thousands or millions of documents and present it to a user. In interactive scenarios, such as when organizing the results of a search (e.g., on research papers), the reading tree will be generated on the fly for a relatively
reasonable number of documents (e.g., top 200). In these interactive scenarios, our approach is very efficient.

There may be scenarios where it is desired to compute a reading tree over thousands of documents and store it in order to show parts of it depending on the user or the context. Then such computation can be performed offline. As a trade-off between disk space and time requirements, one may decide to store only the output of the topic model and similarity graph and compute a reading tree on the fly based on the set of documents that a user has selected in an application. For example, for short documents (or snippets) in Figure 6(b), tree generation is still less than 1 minute for 10000 documents.

**D. Reading Tree Evaluation**

1) **Evaluation Metric:** In order to compare and evaluate the structures generated from our algorithms, we need a way to compare trees. Edit distance metrics, initially introduced for string comparison, have been used to compare ordered trees [34]. Ordered labeled trees are trees in which the left-to-right order among siblings is significant. A distance between two trees is computed by considering an optimal mapping between two trees as the minimum cost of a sequence of elementary operations that converts one tree into the other. An alternative to mapping and tree edition is tree alignment [35].

Our reading order problem is different, and thus we are not interested in how identical two trees are. We care for the relative ordering of each pair of documents. To quantify the tree difference based on the pairwise document orderings, we first build the adjacency matrix \( A \) for a tree structure using the following formula:

\[
A_{ij} = \frac{1}{\text{numhops}(d_i \rightarrow d_j)} \quad \text{if there is a directed path from } d_i \text{ to } d_j; \quad \text{otherwise } A_{ij} = 0.
\]

\( A_{ij} \) is the element of the adjacency matrix corresponding to documents \( d_i \) and \( d_j \) and \( \text{numhops}(d_i \rightarrow d_j) \) is the number of hops from document \( d_i \) to \( d_j \).

To measure the difference of two tree structures over a set of documents represented by matrices \( A \) and \( \hat{A} \), we use the mean squared error (MSE), which is defined as:

\[
MSE(A, \hat{A}) = \frac{1}{\text{numdocs}} \sum_{i,j=1}^{\text{numdocs}} (A_{ij} - \hat{A}_{ij})^2
\]

Figure 7 illustrates an example of how to compare tree structures using MSE. In this example, \( A \) is the adjacency matrix of the ground truth, \( \hat{A}_{\text{result1}} \) (\( \hat{A}_{\text{result2}} \)) is the adjacency matrix of the tree result1 (result2). Since we have \( MSE(A, \hat{A}_{\text{result1}}) = 0.2469 > MSE(A, \hat{A}_{\text{result2}}) = 0.1229 \), the tree structure result2 is better than result1 because compared with the ground truth documents are placed mostly in the right order. There are two sources for the mean squared error of result2. We observe that document \( d_1 \)'s relative order is different in result2 compared to the ground truth. In the ground truth, it is \( A(d_1, d_4) = 0.5 \) as the number of hops from document \( d_1 \) to \( d_4 \) is two, while in result2, \( \hat{A}_{\text{result2}}(d_1, d_4) = 1 \) as document \( d_1 \) becomes direct parent node of document \( d_4 \). Moreover, the fact that document \( d_2 \) is a parent node of document \( d_4 \) is completely missed in result2.

2) **Using Wikipedia for Ground Truth:** There is no ground truth on the actual reading order for any set of documents. As one strategy for evaluating our approach, we use the Wikipedia hierarchy of categories to approximate the reading order for Wikipedia pages. For example, “Cluster Analysis” is a subcategory of “Machine Learning”. We assume that all the article pages belonging to “Machine Learning” are more general and should be read before articles in “Cluster Analysis”.

However, Wikipedia’s hierarchy of categories is far from perfect. It contains errors, and cycles. To build our ground truth, we start from the “Machine Learning” category and expand 3 steps away in the category structure building a hierarchy with no cycles. After removing empty categories and articles, we have 118 categories, from which we random select two articles out of all the pages from each category, resulting in 236 articles in total.

We feed these articles to our system and we compare its output to the ground truth using different parameter combinations, \( \tau = \{0.5, 0.7, 0.9\} \), \( \kappa = \{0.001, 0.005, 0.01\} \) and for 20 topics. Table IV summarizes the results. The low MSE suggests that our reading trees generate good reading orders. For \( \tau = 0.7, \kappa = 0.005 \), we obtain the best performance with the minimum MSE = 0.1214.

Our actual algorithm performance may be better than these MSE scores indicate, because the category hierarchy of Wikipedia does not provide a perfect ordering. For example,
“Machine Learning Researchers” is a subcategory of “Machine Learning”. Pages under “Machine Learning” cover the content of machine learning algorithms and applications. However, the pages under “Machine Learning Researchers” cover the background of the person, personal history, etc.

3) Using Experts for Ground Truth Creation: As a different evaluation strategy, we also asked experts to generate the ground truth for different topics (Data Mining, Biology, Math, and Arts) and sets of Wikipedia pages. For each set of documents, we asked the opinion of two experts. Of course, it is not possible to ask experts to create very large graphs, so each tree created by an expert is around 20 documents. Also we observed that on the same set of documents, two experts may create different trees. This is reasonable, since there is an amount of subjectivity in the problem. Our purpose is not to create the exact same tree but create the same reading orders as possible by $MSE$.

We then compared our reading trees against the experts’ ground truth. Since for each set we had two trees as ground truth, we compared the tree we generated to each one of them and then take the average $MSE$ as the final $MSE$ for a generated tree. The average $MSE$ across the four topics is 0.18 for $s = 10$, $\tau = 0.9$, $\kappa = 0.001$. We observed that the Math tree had the poorer performance ($MSE=0.3$). We believe that in Math other types of document relationships, e.g. pre-requisites, may be more frequent than specificity relations.

4) User study: A third way to evaluate the results is to have a user study, where feedback is collected from users that evaluate the generated reading trees manually. This evaluation method is effective only when the size of the tree is small, and is hard to utilize when the output tree structure is large.

We created user studies for three scenarios: (a) research scenario, where the purpose is to see research papers in some logical order, (b) news editor scenario, where the tree is presented to an editor for selecting articles to place in a news publication and (c) search scenario, where the purpose is to search pages related on a topic. For each scenario, we use $DB$, $NEWS$, and the $WIKIPEDIA$ pages for data mining, respectively, and we pre-generated the reading graphs for all the documents per case. We intentionally used the same configuration, $s = 40$, $\tau = 0.9$, $\kappa = 0.001$, for all datasets because we wanted to see how good a single configuration will be for different data.

For scenario (a), we asked 3 CS researchers and 2 students, for (b) we asked 2 editors, and for (c) we asked 10 non-CS researchers. For each scenario, we selected random subtrees from the graph containing at most 15 documents, and we asked the users to evaluate them by counting how many documents they thought were in the wrong order. Each user evaluated 4 trees and each tree was evaluated by one user. The tree was represented graphically. Each node contained a short description for each document in the node. Clicking the description links opened the whole document allowing the user to inspect the document.

The average number of misplaced documents per tree reported was: 2.2 ($DB$), 2.9 ($NEWS$), and 2.5 ($WIKIPEDIA$). The respective percentages over the document set size were: 14.6% ($DB$), 19.33% ($NEWS$), and 16.66% ($WIKIPEDIA$). We observed that the percentage was higher for the $NEWS$. It is likely that the selected configuration was not optimal for this collection. Still, the result were useful for the editors.

E. Topic Model Calibration

As a final note, we would like to discuss the effectiveness of the score propagation method we use for the topic model calibration. The topic model is a probabilistic model and hence every time it is executed over exactly the same inputs, its output may be different. In this paper, we utilize the topic model calibration to compensate for the errors and variations caused by the topic model. In order to justify its effectiveness, we compare the trees generated with and without score propagation.

For this purpose, we ran the tree generation algorithm 11 times with all meaningful parameter combinations of $s = \{5, 10, 15, 20, 25, 30, 35, 40\}$, and $\kappa = \{0.01, 0.03, 0.05, 0.07, 0.09, 0.11, 0.13, 0.15\}$. We compare the tree generated in one of the last 10 runs with the tree generated in the first run (which serves as a kind of ground truth). Figure 8(a) shows the average $MSE$ score and Figure 8(b) shows the worst. Each point corresponds to one of the 10 repetitions (shown on the x-axis) and is the average $MSE$ value of all parameter combinations (shown on the y-axis).

With score propagation, both the worst and the average performance of the tree generation method are better. The score propagation acts as a smoothing filter compensating for the variations in the output of the topic model.
In this paper, we proposed generating reading trees to organize a collection of documents from general to more specific content. We proposed a set of algorithms for topic model calibration and tree generation. We evaluated the impact of the various parameters of the problem for the tree form and the performance of the approach. As there is no ground truth for our problem, we applied different methods for judging the result of the reading tree generation.

As future work, we would like to examine methods for incrementally growing and refining a reading tree based on a subset of known documents. Other future research directions include considering other types of document relationship for document sequencing, and personalizing reading trees for different users.

REFERENCES

VIII. CONCLUSION AND FUTURE WORK
In this paper, we proposed generating reading trees to organize a collection of documents from general to more specific content. We proposed a set of algorithms for topic model calibration and tree generation. We evaluated the impact of the various parameters of the problem for the tree form and the performance of the approach. As there is no ground truth for our problem, we applied different methods for judging the result of the reading tree generation.

As future work, we would like to examine methods for incrementally growing and refining a reading tree based on a subset of known documents. Other future research directions include considering other types of document relationship for document sequencing, and personalizing reading trees for different users.