Mining Evolutionary Multi-Branch Trees from Text Streams

Xiting Wang[‡§], Shixia Liu[§*] Yangqiu Song[♮], Baining Guo[§‡]
[‡]Tsinghua University, Beijing, China
[§]Microsoft Research Asia, Beijing, China
[♮]Hong Kong University of Science and Technology, Hong Kong
{v-xitwan,shixia.liu}@microsoft.com,yqsong@cse.ust.hk, bainguo@microsoft.com

ABSTRACT

Understanding topic hierarchies in text streams and their evolution patterns over time is very important in many applications. In this paper, we propose an evolutionary multi-branch tree clustering method for streaming text data. We build evolutionary trees in a Bayesian online filtering framework. The tree construction is formulated as an online posterior estimation problem, which considers both the likelihood of the current tree and conditional prior given the previous tree. We also introduce a constraint model to compute the conditional prior of a tree in the multi-branch setting. Experiments on real world news data demonstrate that our algorithm can better incorporate historical tree information and is more efficient and effective than the traditional evolutionary hierarchical clustering algorithm.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Knowledge acquisition; G.3 [**Probability and Statistics**]: [Time series analysis]

Keywords

Time Series Data, Multi-Branch Tree, Topic Evolution, Visualization, Clustering

1. INTRODUCTION

With an increasingly large number of textual documents (e.g., news, blogs) published on the Web every day, there is an increasing need to better understand the topics in a text stream. In many applications, topics are naturally organized in a hierarchy and the hierarchy often evolves over time [9]. Consequently, there have been some initial efforts to model such evolving hierarchies. The state-of-the-art approach, evolutionary hierarchical clustering [9], aims to generate evolving binary trees to organize the topics at different times. However, they may fail to provide interpretable topic results since most of the topic trees in real world applications are not binary [8]. It is therefore important to effectively learn an evolving multi-branch tree representation, providing users a coherent view of content transitions.

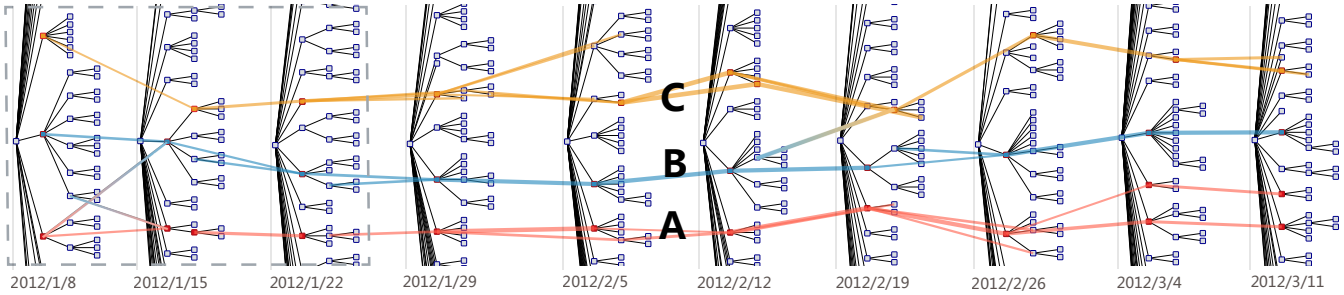
*S. Liu is the correspondence author.

KDD'13, August 11–14, 2013, Chicago, Illinois, USA.

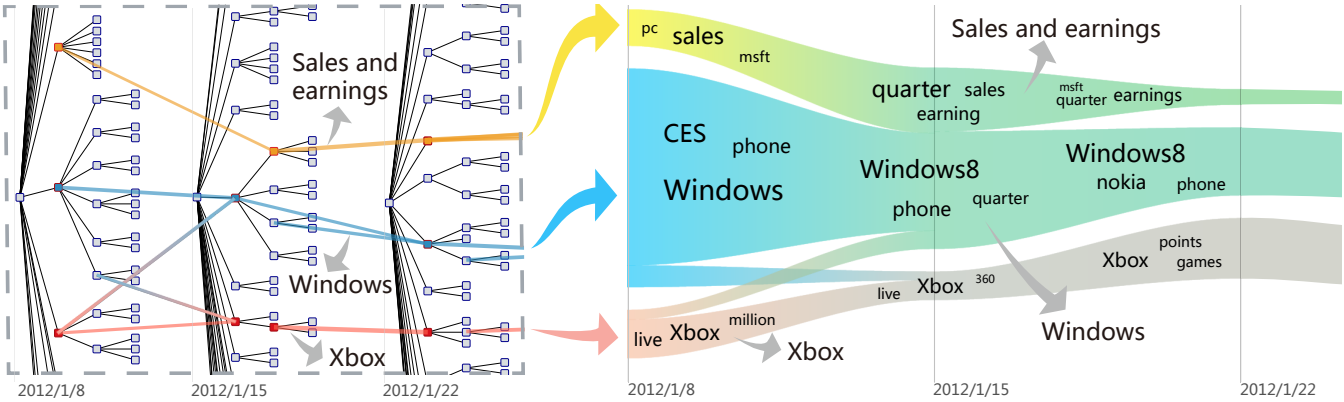
In this paper, we define and study the problem of mining evolving multi-branch topic trees inside a text stream, as well as their evolution patterns over time. Specifically, we take a news dataset as an example to illustrate the basic idea. Fig. 1(a) shows part of the evolving topics and their hierarchical structures extracted from this dataset. The three labeled topics are "xbox"(A), "windows"(B), and "sales and earnings"(C) from Jan. 8 to Mar. 11, 2012. We align the correlated topics across different trees according to their content similarity. From the alignment edges, we can see the three topics are quite stable during this time period, with a few splitting/merging relationships between them over time. With such evolutionary trees and their visual representation in Fig. 1(a), users can easily examine: 1) the **evolution of multi-branch trees** and their **content alignment** over time; 2) the topics of interest and their **evolving patterns** (e.g., splitting/merging) at different levels of these trees.

To better understand the evolving patterns of user-selected topics, we leverage a dynamic topic visualization technique, TextFlow [10], to illustrate the topic merging/splitting patterns. In this visualization, a river flow metaphor is adopted to illustrate topic evolution over time (Fig. 1(b)). Each colored layer represents a topic. The varying layer height along the horizontal axis represents the number of documents for the topic at each time point. Like a river flow in the real world, the topic flow can either be split into several branches when the corresponding topic splits, or combined with several other branches into one layer when the corresponding topics merge together. Fig. 1(b) shows the splitting/merging patterns of topics "xbox," "windows," and "sales and earnings" from Jan. 8 to 28. Topics "windows" and "sales and earnings" merge in the week of Jan. 15 when Microsoft reported its quarterly revenue. To discover more information about the merging, we browse the related news. Some news items report on the "sales and earnings" of "windows." For example, one of the articles has the title "Windows sales slowdown as Microsoft reports Q2 revenue up 5%." These two topics split in the next week as the association becomes weaker. Another interesting pattern is that part of the "windows" topic splits itself from the main topic in the first week and then joins "xbox" in the next week. The major reason is that Dave Culter, the father of Windows NT, shifted his focus to Xbox and was working "to extend xbox beyond its status as a gaming platform."

Motivated by this example, we aim to generate a sequence of coherent multi-branch topic trees. Each tree in the sequence should be similar to the one at the previous time point (smoothness). It also needs to well describe the document distribution at that time point (fitness). However, it is quite challenging to achieve the desired results. First, it is not trivial to generate evolving multi-branch tree representations as well as to model their evolution patterns over time. Although the state-of-the-art multi-branch clustering methods [8, 16] can generate a topic tree with a high fitness value, they cannot



(a) Ten evolving trees from Jan. 8 to Mar. 17 and the highlighted topics "xbox"(A), "windows"(B) and "sales and earnings"(C)



(b) Splitting/merging patterns of topics "xbox," "windows," and "sales and earnings" (Jan. 8 to Jan. 28) and the transition from the tree representation to the TextFlow visualization. With this visualization, the evolving splitting/merging relationships are clearly conveyed.

Figure 1: Evolutionary trees and patterns in Bing news data. 66,528 news articles were collected from Bing News using query word "microsoft." These articles were gathered from Jan. 8, 2012 to Jul. 21, 2012. We grouped the data by week over the 28-week period. Accordingly, 28 trees were generated. The average tree depth was 4, the average internal node number was 99, and the average node number of the first level was 21. Here we select part of the evolving trees from Jan. 8 to Mar. 17.

guarantee the smoothness between topic trees. One way to solve this problem is to minimize the tree distance difference between two correlated node pairs at two consecutive time points. This method can improve the smoothness between trees to some extent. However, it may fail to reconstruct an optimized tree for the current time point since the parent-child relationships are lost. Second, online documents (e.g., news articles) arrive regularly and thus they are usually large in number. Since the complexity of tree-based algorithms are non-linear to the data number, it is therefore very time-consuming to generate a sequence of topic trees that well balances the fitness of each tree and the smoothness between adjacent trees.

To tackle the above challenges, we propose an algorithm, an evolutionary Bayesian rose tree (EvoBRT), to automatically learn tree evolutionary patterns. In our work, a Bayesian rose tree (BRT) [8] is adopted to handle the multi-branch tree construction problem, as well as to maintain a high fitness for human understanding. To preserve the smoothness between adjacent trees, we formulate the evolutionary clustering problem as a Bayesian online filtering algorithm inspired by the method in [5]. A tree prior is introduced in the BRT learning framework, which is formulated as a Markov random field (MRF). The key here is to define the energy function of the MRF model to measure the smoothness cost. A previous study [19] has shown that a tree can be uniquely defined by a set of triples and fans. A triple is a sub-tree with three leaf nodes and two internal nodes (Fig. 3(a)), while a fan is a sub-tree with three leaf nodes and one internal node (Fig. 3(b)). We argue that, in order to create a sequence of coherent topic trees, we need to keep as many sub-trees as possible when generating a new tree. To this end, we define the smoothness cost as proportional to the number of violated triples/fans between the adjacent trees. Our experiments show that the triple- and fan-based measures work better than the evolutionary hierarchical clustering algorithm [9] in preserving smoothness between trees.

The computational complexity of our algorithm is mainly determined by two parts: the construction of a Bayesian rose tree and the parametrization of the tree prior given a pre-defined tree. We leverage our previous work [16] to generate the Bayesian rose tree, which reduces the complexity from $O(n^2 \log(n))$ to $O(n \log(n))$. The complexity of the tree prior parametrization is mainly caused by calculating the number of violated triples/fans. Directly computing the number usually takes $\Omega(n^3)$ time, which is very time consuming. To solve this problem, we build a constraint tree from all the triples and fans. By leveraging this tree index, we reduce the calculation time to $O(n \log(n))$.

2. RELATED WORK

2.1 Constrained Hierarchical Clustering

In the area of data mining, researchers have developed various approaches to perform constrained hierarchical clustering. Based on the constraint type, they can be classified into two categories: pairwise approaches and triplewise approaches. Pairwise approaches incorporate the constraints in the form of must-links and cannot-links, which indicate that two samples must or cannot be in the same cluster [11, 18]. Since must-links and cannot-links do not consider hierarchical information, these methods may fail to characterize the hierarchical document distribution.

Triplewise approaches incorporate triple constraints (e.g., two samples must be combined before the other sample is combined with either of them) among the data to generate clusters. Existing methods consider two different ways to use triple constraints, *metric-based* approaches and *instance-based* approaches [4]. Metric-based approaches learn a distance or similarity metric from the constraints and then embed the metric in the clustering process [4, 13, 21, 30]. Instance-based approaches follow all triplewise constraints in the bottom-up merging process and will fail to generate a hierarchy if one of them is violated [4, 15, 29].

In contrast to the above algorithms, which are designed for building a static binary tree, our algorithm aims to generate evolving multi-branch hierarchies. A multi-branch tree contains both triples and fans, so the existing approaches do not work since they cannot handle fans. In addition, to model evolving patterns, our algorithm automatically generates constraints for the tree at time t based on the tree structure at $t-1$, while the constraints in constrained hierarchical clustering are predefined.

2.2 Evolutionary Topic Analysis

Recent efforts in topic analysis have focused on developing advanced machine learning algorithms to extract evolving topics, such as dynamic latent Dirichlet allocation and its variations [1, 6], and hierarchical Dirichlet processes [2, 3, 25, 26, 28]. The evolving topics may be correlated with others by various relationships over time. The most intuitive relationships are *topic correlation* [23] and *common topics* [24]. Recently, TextFlow was developed to help users analyze topic evolution patterns, including topic birth, death, splitting, and merging, in text data [10, 12]. However, none of the above methods focus on mining evolving trees.

A previous study has shown that it is very useful for information understanding and consumption if users are provided with hierarchical topic information over time [27]. However, how to efficiently and effectively mine the hierarchical topics as well as their evolving patterns has not been solved yet. The most related method to ours is the evolutionary hierarchical clustering algorithm [9]. It measures the difference between trees by the average distance between all node pairs. However the tree distance metric is not sufficient to reconstruct a tree and measure the smoothness between trees since the parent-child relationships are lost. In contrast to evolutionary hierarchical clustering, we introduce two constraints, triples and fans, into the model to guarantee high smoothness between topic trees. In addition, we also choose the Bayesian rose tree [8] as our base representation to discover multi-branch structures in text data instead of binary tree structures.

3. BACKGROUND

To perform evolutionary hierarchical clustering, we adopt the static multi-branch tree as the base representation. In this section, we briefly introduce its definition and construction method.

Given a set of text documents $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, where each document is represented as a feature vector $\mathbf{x} \in \mathcal{R}^{|\mathcal{V}|}$ and $|\mathcal{V}|$ is the vocabulary size of the corpus, a multi-branch tree either consists of a single leaf $\mathbf{x} \in \mathcal{D}$, or consists of a set of sub-trees $T_1, T_2, ..., T_{n_T}$ whose parent is the root node T. Each sub-tree T_i is also defined in the same way. Here n_T can be larger than two.

To infer the multi-branch tree structure, we follow the Bayesian rose tree (BRT) [8] approach, which greedily estimates the tree structure based on probability $P(\mathcal{D}|T)$. Initially, each document is regarded as an individual tree on its own: $T_i = \{\mathbf{x}_i\}$. Then the algorithm repeatedly selects two trees T_i and T_j and combines them into a new tree T_m by a *join*, *absorb*, or *collapse* operation (Fig. 2),

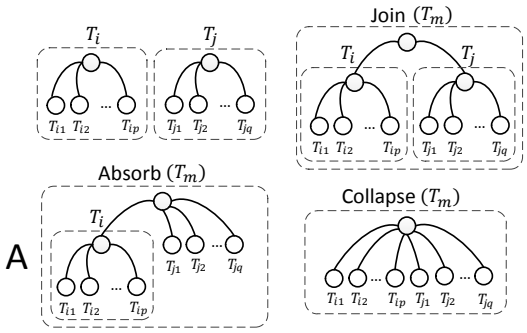


Figure 2: Basic operations of BRT.

aiming to maximize the ratio of probability:

$$\frac{p(\mathcal{D}_m|T_m)}{p(\mathcal{D}_i|T_i)p(\mathcal{D}_j|T_j)}, \tag{1}$$

where $p(\mathcal{D}_m|T_m)$ is the likelihood of $\mathcal{D}_m$ given the tree T_m, $\mathcal{D}_m = \mathcal{D}_i \cup \mathcal{D}_j$ represents all the data under T_m. $p(\mathcal{D}_m|T_m)$ is defined as:

$$p(\mathcal{D}_m|T_m) = \pi_{T_m} f(\mathcal{D}_m) + (1 - \pi_{T_m}) \prod_{T_i \in children(T_m)} p(\mathcal{D}_i|T_i), \tag{2}$$

where $f(\mathcal{D}_m)$ is the marginal probability of $\mathcal{D}_m$, and $children(T_m)$ is the child set of T_m. π_{T_m} is the prior probability that all the data in T_m is kept in one cluster. Specifically, it is defined as:

$$\pi_{T_m} = 1 - (1 - \gamma)^{n_{T_m} - 1}, \tag{3}$$

where n_{T_m} is the child number of T_m, and $0 \leq \gamma \leq 1$ is the hyper-parameter to control the partition granularity.

To represent the marginal distribution $f(\mathcal{D})$, we use the DCM distribution [16, 17]

$$f_{DCM}(\mathcal{D}) = \prod_i^n \frac{m_i!}{\prod_j^{|\mathcal{V}|} x_i^{(j)}!} \cdot \frac{\Delta(\boldsymbol{\alpha} + \sum_i \mathbf{x}_i)}{\Delta(\boldsymbol{\alpha})}, \tag{4}$$

where $m_i = \sum_j^{|\mathcal{V}|} x_i^{(j)}$, and $\boldsymbol{\alpha} = (\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(|\mathcal{V}|)})^T \in \mathcal{R}^{|\mathcal{V}|}$ is the parameter that controls the Dirichlet distribution, which is the prior of the multinomial distribution of each cluster.

4. EVOLUTIONARY TREE MODELING

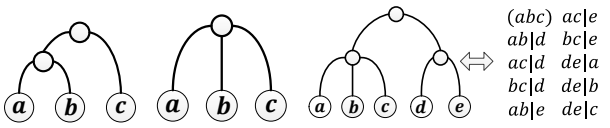
In this section, we first introduce the overall procedure of evolutionary tree construction. Then we illustrate the formulation of constraint modeling and related operations.

4.1 Algorithm Overview

In our model, we assume text data comes in a sequential and continuous way. At each time t, we have a set of documents, which is denoted as $\mathcal{D}^t = \{\mathbf{x}_1^t, \mathbf{x}_2^t, ..., \mathbf{x}_{n_t}^t\}$, where n_t is the number of documents coming at that time point. We assume there is an underlying tree T^t that organizes the documents at t. To capture the tree structure changes behind the data, we formulate the learning procedure as a Bayesian online filtering process:

$$p(T^t|\mathcal{D}^t, T^{t-1}) \propto p(\mathcal{D}^t|T^t)p(T^t|T^{t-1}). \tag{5}$$

With this formulation, the new tree T^t depends on both the likelihood of the current data $p(\mathcal{D}^t|T^t)$ and conditional prior $p(T^t|T^{t-1})$ that measures the difference between T^t and T^{t-1}. Accordingly, our model considers both the fitness and historical smoothness costs as in the evolutionary hierarchical clustering algorithm [9]. For simplicity, here we only use one-step historical smoothness. It is also



(a) Triple: $ab|c$ (b) Fan: (abc) (c) Tree representation

Figure 3: Tree representation by triples and fans.

easy to incorporate more than one historical tree in the conditional prior $p(T^t|\mathcal{D}^t, T^{t-1}, T^{t-2}, ...)$.

Similar to BRT, directly maximizing $p(\mathcal{D}^t|T^t)p(T^t|T^{t-1})$ is intractable, since there are a super-exponential number of candidate trees for T^t. We then follow the greedy construction method of BRT [8] to select two sub-trees and one of the three types of combinations (join, absorb, and collapse) to construct a larger (sub-)tree. The selection aims to maximize the following posterior test ratio:

$$\frac{p(\mathcal{D}^t_m|T^t_m)p(T^t_m|T^{t-1})}{p(\mathcal{D}^t_i|T^t_i)p(T^t_i|T^{t-1}) \cdot p(\mathcal{D}^t_j|T^t_j)p(T^t_j|T^{t-1})}, \tag{6}$$

where T^t_i and T^t_j are two candidate sub-trees that are considered for generating T^t_m using the three types of combinations, D^t_i and D^t_j are the corresponding document sets, and $D^t_m = D^t_i \bigcup D^t_j$.

By defining the energy function, which measures the smoothness cost of merging T^t_i and T^t_j given T^{t-1}, as

$$V_{T^{t-1}}(\{T^t_i, T^t_j\} \rightarrow T^t_m), \tag{7}$$

we can formulate the conditional tree prior in a recursive way:

$$p(T^t_m|T^{t-1}) = p(T^t_m|T^t_i, T^t_j, T^{t-1})p(T^t_i|T^{t-1})p(T^t_j|T^{t-1}), \tag{8}$$

where

$$p(T^t_m|T^t_i, T^t_j, T^{t-1}) \triangleq \frac{1}{Z}\exp\Big(-\lambda V_{T^{t-1}}(\{T^t_i, T^t_j\} \rightarrow T^t_m)\Big), \tag{9}$$

and λ is the constraint weight that balances the importance of smoothness and tree likelihood. Inspired by a simpler Markov random field (MRF) defined on flat clusters in [5], we regard the conditional prior $p(T^t|T^{t-1})$ as a Gibbs distribution based on a recursively defined MRF. The energy function of the MRF can be parameterized as a sum of a set of $V_{T^{t-1}}(\{T^t_i, T^t_j\} \rightarrow T^t_m)$. With this parametrization, we rewrite Eq. (6) as:

$$\frac{p(\mathcal{D}^t_m|T^t_m)}{p(\mathcal{D}^t_i|T^t_i)p(\mathcal{D}^t_j|T^t_j)} \cdot \frac{1}{Z}\exp\Big(-\lambda V_{T^{t-1}}(\{T^t_i, T^t_j\} \rightarrow T^t_m)\Big). \tag{10}$$

Since the first term corresponds to the BRT algorithm (Eq. (1)), the key is then to calculate the second term, the smoothness cost $V_{T^{t-1}}(\{T^t_i, T^t_j\} \rightarrow T^t_m)$, which is illustrated in the next sub-section.

4.2 Constraint Modeling

The major goal of the smoothness cost is to preserve as many common tree structures as possible. To this end, we first introduce the triple- and fan-based constraints and use them to measure the cost. Then a constraint tree is built for efficient computation. Finally, we illustrate how to gradually modify the constraint tree for better measuring the cost.

4.2.1 Triple, Fan, and Constraint Tree

We first introduce some preliminary definitions that are useful for subsequent discussions.

Definition 1. *A **triple** (Fig. 3(a)) is a sub-tree with three leaf nodes and two internal nodes. We denote it as **ab|c** where **a** and **b** are the two closest leaf nodes and **c** is the third leaf node.*

Definition 2. *A **fan** (Fig. 3(b)) is a sub-tree with three leaf nodes and one internal node. We denote it as **(abc)** where **a**, **b**, and **c** are the three leaf nodes.*

Binary trees only contain triples, while multi-branch trees contain both triples and fans. The example in Fig. 3(c) illustrates the relationship between a multi-branch tree and its corresponding triples/fans. This tree contains nine triples and one fan.

The following lemma illustrates the relationship between a multi-branch tree and its related triples/fans, which was proposed by Ng et al. [19].

Lemma 1. *A multi-branch tree T can be uniquely defined by a set of triples and fans.*

This lemma indicates that triples and fans contain all the hierarchical information of a multi-branch tree. Since a tree can be uniquely reconstructed by a set of triples and fans, we measure the smoothness cost by the violated triples/fans between adjacent trees. A tree with n leaves contains C^3_n triples and/or fans. Thus, directly computing the violation number usually takes $O(n^3)$ memory and $\Omega(n^3)$ time, which is very time- and memory-consuming. To solve this problem, we build a tree to organize all the related triples and fans. We call it a *constraint tree*.

Definition 3. *A **constraint tree** $\tilde{T}^t$ hierarchically organizes the triples/fans inferred from $\mathcal{D}^t$. It is initialized based on the previous tree T^{t-1}, and modified as the corresponding triples/fans are violated.*

In the next three sub-sections, we illustrate the constraint tree initialization and modification.

4.2.2 Constraint Tree Initialization

The basic idea of initializing the constraint tree $\tilde{T}^t$ is to map each document in $\mathcal{D}^t$ to its most relevant topic in T^{t-1}. For a document that does not belong to any topic in T^{t-1}, a new topic is then generated at t. In our implementation, we propose two alternative measures to compute the similarity between $\mathbf{x}^t_i$ and $\mathcal{D}^{t-1}_m$. The first is based on the cosine similarity between documents:

$$sim_{cos}(\mathbf{x}^t_i, \mathcal{D}^{t-1}_m) \triangleq \cos(\mathbf{x}^t_i, \sum_{\mathbf{x}^{t-1}_j \in \mathcal{D}^{t-1}_m} \mathbf{x}^{t-1}_j). \tag{11}$$

An alternative measure, the prediction measure, is based on the conditional probability of $\mathbf{x}^t_i$ given $\mathcal{D}^{t-1}_m$ [7]:

$$sim_{pred}(\mathbf{x}^t_i, \mathcal{D}^{t-1}_m) \triangleq \log p(\mathbf{x}^t_i|\mathcal{D}^{t-1}_m) = \log \int_\theta p(\mathbf{x}^t_i|\theta)p(\theta|\mathcal{D}^{t-1}_m)d\theta. \tag{12}$$

We adopt a greedy method to map the document $\mathbf{x}^t_i$ to the sub-trees, which searches T^{t-1} in a top-down manner. We compare the similarity value of a parent with those of its children by leveraging one of the above two similarity measures. If the similarity value of the parent is larger than any of its children, as well as a pre-defined threshold s_0, we stop the search process and set the parent as the most relevant topic; otherwise we treat the child with the highest relevance value as the new parent and repeat the process.

After mapping all the data $\mathbf{x}^t_i$ in $\mathcal{D}^t$ to the possible topics in T^{t-1}, we have a new tree $\tilde{T}^t$, which is the initialization of the constraint tree. With this initial constraint tree, one way to compute the smoothness cost is as follows: When building a new tree T^t, we compute how many triples and fans are violated when combining two candidate sub-trees T^k_i and T^k_j by leveraging the constraint tree. Although this method is intuitive, it has one major problem. This is due to the fact that we ignore conflicting constraints. For example, given a

constraint tree $\tilde{T}^{t-1}$ in Fig. 4, if we combine a and d, two types of constraint states will be introduced. The first type is the violated constraints. In this example, three triples/fans are violated. Here we take the triple $ab|d$ as an example. This triple indicates that a and b should be combined first and then they are combined with d. However, if a and d are combined first, this triple is violated. The second type is the conflicting constraints, which are the triples/fans that cannot co-exist in the constraint tree. Considering fan (abc), if a and d are combined, (abc) conflicts with the triple $bc|d$ since they cannot both be in this tree.

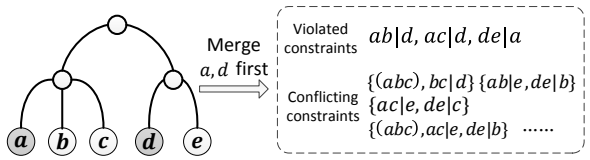


Figure 4: Two types of constraint states.

Besides the violated constraints, the conflicting constraints also influence the smoothness cost. However, it is not easy to compute the cost caused by conflicting constraints since the relationships between them are complicated. For example, one constraint may conflict with multiple constraints. Moreover, besides the pairwise conflicts, there are triplewise and multiple conflicts. As shown in Fig. 4, if a and d are combined, then (abc), $ac|e$, and $de|b$ conflict even though any two of them do not conflict with each other. As a result, it is hard to measure the corresponding cost even if we list all the conflicting constraints. To tackle this issue, we introduce two basic operations, MERGE and SPLIT, to the constraint tree.

4.2.3 Basic Operations and Their Cost

To better measure the cost introduced by conflicting constraints, we define two basic operations, MERGE and SPLIT. Fig. 5 briefly illustrates the two operations.

DEFINITION 4. **MERGE**: *a sub-tree $\tilde{T}_k$ forwards the data of its own and its children to its parent $\tilde{T}_l$. Then $\tilde{T}_k$ itself is removed from the constraint tree.*

DEFINITION 5. **SPLIT**: *some children of a sub-tree $\tilde{T}'_l$ redirect their parent to a newly generated child $\tilde{T}_k$ of $\tilde{T}'_l$.*

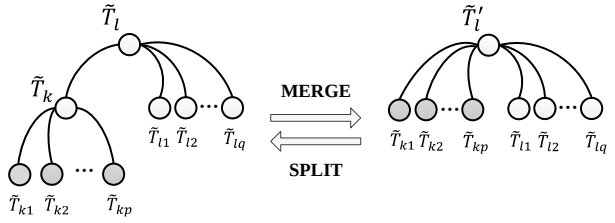


Figure 5: MERGE and SPLIT operations on the constraint tree.

Next, we illustrate the major reason why these two operations are enough to remove the conflicts from a constraint tree and make it consistent. For simplicity, we take a two-level tree as an example to illustrate the basic idea. As shown in Fig. 2, BRT provides three types of combinations, a *join*, an *absorb*, and a *collapse*. If sub-trees T_i and T_j are combined with a BRT operation that is different from the one in the constraint tree, conflicting constraints will be introduced. The basic idea of avoiding conflicts is to update the constraint tree to make it consistent with the current data organization. For example, assume sub-trees T_i and T_j are combined together by an absorb operation (Fig. 2A). If they appear in the constraint tree as shown in Fig. 6B, conflicting constraints are introduced. To solve them, we change the constraint tree into the one in Fig. 6C using a MERGE operation. More examples of using the MERGE/SPLIT operation(s) to modify the constraint tree are shown in Fig. 6, which demonstrate that the changes between the constraint trees that correspond to the BRT operations can all be handled with the SPLIT or MERGE operation(s). Consequently, they are enough to maintain a consistent constraint tree.

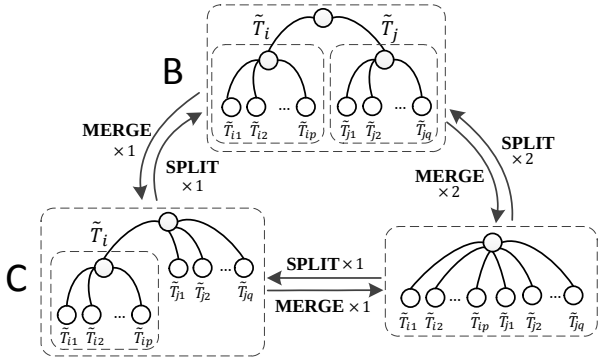


Figure 6: Update constraint trees by MERGE/SPLIT.

With MERGE/SPLIT, we can then measure the cost from conflicting constraints by counting the violated constraints in the two operations. The calculation method is given by the following theorems. We take the MERGE/SPLIT operation in Fig. 5 as an example in the following discussions.

THEOREM 1. *In a* MERGE *operation, only triples may be violated, and violated triples become fans. The number of violated triples is*

$$V_{\text{MERGE}}(\{\tilde{T}_k, \tilde{T}_l\} \to \tilde{T}'_l) = \frac{|\tilde{\mathcal{D}}_k|^2 - \sum |\tilde{\mathcal{D}}_{ki}|^2}{2}(|\tilde{\mathcal{D}}_l| - |\tilde{\mathcal{D}}_k|), \tag{13}$$

where $|\tilde{\mathcal{D}}_k|$ is the number of leaves in the tree $\tilde{T}_k$.

PROOF. Given a (sub-)tree T, the number of fans is

$$|\mathcal{F}_T| = \sum_{\substack{T_s \in sub-trees(T) \\ |children(T_s)|>2}} \sum_{\substack{T_{si}, T_{sj}, T_{sl} \in \\ children(T_s) \\ T_{si} \neq T_{sj} \neq T_{sl}}} |\mathcal{D}_{si}||\mathcal{D}_{sj}||\mathcal{D}_{sl}|, \tag{14}$$

where $\mathcal{F}_T$ is the set of fans in T, and $|\mathcal{F}_T|$ is the number of fans in the set.

Since all the triples in $\tilde{T}'_l$ are contained in $\tilde{T}_l$, the MERGE operation only violates triples and changes them to fans. Thus the cost of the MERGE operation is the difference in fan number between $\tilde{T}_l$ and $\tilde{T}'_l$.

$$|\mathcal{F}_{\tilde{T}'_l}| - |\mathcal{F}_{\tilde{T}_l}| = \frac{|\tilde{\mathcal{D}}_k|^2 - \sum |\tilde{\mathcal{D}}_{ki}|^2}{2}(|\tilde{\mathcal{D}}_l| - |\tilde{\mathcal{D}}_k|). \tag{15}$$

□

Similarly, we can prove the following theorem on the SPLIT operation.

THEOREM 2. *In a* SPLIT *operation, only fans may be violated, and the violated fans become triples. The number of violated fans is:*

$$V_{\text{SPLIT}}(\tilde{T}'_l \to \{\tilde{T}_k, \tilde{T}_l\}) = \frac{|\tilde{\mathcal{D}}_k|^2 - \sum |\tilde{\mathcal{D}}_{ki}|^2}{2}(|\tilde{\mathcal{D}}_l| - |\tilde{\mathcal{D}}_k|). \tag{16}$$

4.2.4 Constraint Tree Modification and Its Cost

In Sec. 4.2.3, we assume the sub-trees to be combined are adjacent to each other. In real applications, however, the two sub-trees may not be adjacent. In such a case, we first move them to the closest common ancestor by a sequence of MERGE operations. Once they are under the same ancestor, we perform the MERGE/SPLIT operation(s) to make the related constraint tree consistent. As a result, the smoothness cost introduced by the conflicts can be calculated by summing up the violation costs of all MERGE/SPLIT operations:

$$V_{T^{t-1}}(\{T_i^t, T_j^t\} \to T_m^t) = \sum_o V_{\text{OPT}_o}, \tag{17}$$

where OPT_o is a MERGE operation or a SPLIT operation.

5. COMPUTATIONAL COMPLEXITY

The complexity of EvoBRT is determined by three parts: construction of the Bayesian rose tree, constraint tree initialization, and computation of the violated constraints. The complexity results are summarized in Table 1.

Several methods have been proposed to implement BRT [16]. In this paper, we leverage them to build the multi-branch tree. We use EvoBRT to represent the implementation based on the original BRT [8], and *K*NN-EvoBRT and SpillTree-EvoBRT to represent the ones based on the two approximation algorithms, *K*NN-BRT and SpillTree-BRT [16]. For simplicity, we call these two algorithms the evolutionary approximation algorithms. The related complexity results are shown in Table 1. Interested readers are referred to [16] for a detailed analysis.

To build the initial constraint tree $\tilde{T}^t$, we map each document in $\mathcal{D}^t$ to a proper node in T^{t-1}. Since we search for the best node in a top-down manner, the complexity is $O(nC_V \log n)$.

The complexity of calculating the violated constraints is mainly caused by two parts: counting the violated constraints and updating the posterior test ratios. The first part has $O(n^2 h)$ time complexity in EvoBRT, and $O(nKh)$ time complexity in the evolutionary approximation algorithms. Here h is the depth of tree $\tilde{T}^t$, and K is the number of nearest neighbors. When the constraint tree is modified, some posterior test ratio values will be changed. Since the posterior test ratios are stored in a sorted list, the complexity of the second part is therefore caused by updating this list. The complexity is $O(n^2 h \log n)$ for EvoBRT and $O(nKh \log n)$ for the evolutionary approximation algorithms. The $\log n$ factor is due to updating one value in the sorted list, while $n^2 h$ and nKh are the number of maximum updated values for EvoBRT and the evolutionary approximation algorithms.

Table 1: Time Complexity. C_V is the number of non-zero elements in the vector. The time for initializing $\tilde{T}^t$ is $O(nC_V \log n)$.

	Tree Construction	Violation Calculation
EvoBRT	$O(n^2 C_V + n^2 \log n)$	$O(n^2 h \log n)$
*K*NN-EvoBRT	$O(n^2 C_V + n^2 \log K)$	$O(nKh \log n)$
SpillTree-EvoBRT	$O(ndC_V + nd \log n \log K)$	

6. EXPERIMENTS

To demonstrate the performance of our algorithm, we have conducted a series of experiments on several real datasets. In this section, we first introduce the baseline algorithm. Then the effectiveness of our constraint model is demonstrated by using the 20 newsgroups dataset[1]. Next, we illustrate how our algorithm can well preserve both fitness (likelihood) and smoothness. Finally, we show that our EvoBRT algorithm is as efficient as the BRT algorithm and its variations in [16]. The results show that our algorithm outperforms the baseline in all aspects that we have compared.

[1]http://qwone.com/ jason/20Newsgroups/

In our experiments, we exploited *K*NN-EvoBRT ($K = 50$) due to its efficiency and relative stable performance [16]. For each experiment, the rose tree parameters with the highest likelihood value were selected through a grid search.

6.1 Baseline algorithm

We implemented a baseline algorithm based on the evolutionary hierarchical clustering algorithm [9]. Since this method focuses on binary hierarchies, we only compared its constraint model. More exactly, we incorporated its smoothness function into our framework. The smoothness is defined as

$$\log p_{Dist}(T^t|T^{t-1}) \triangleq -\lambda E_{\substack{r,s\in leaves(T^t)\\ r\neq s}}(d_{T^t}(r,s) - d_{\tilde{T}^t}(r,s))^2, \tag{18}$$

where $d_T(r,s)$ is the tree distance between r and s, and λ is the constraint weight that balances smoothness and tree likelihood.

Here we choose the *squared* heuristic [9] to maximize the smoothness since it can be easily adapted to the multi-branch scenario. With this heuristic, the smoothness cost of combining sub-trees T_i^t and T_j^t is given by

$$V_{(Dist)T^{t-1}}(\{T_i^t, T_j^t\} \to T_m^t) \triangleq E_{\substack{r\in leaves(T_i^t)\\ s\in leaves(T_j^t)}}(d_{T_m^t}(r,s) - d_{\tilde{T}^t}(r,s))^2. \tag{19}$$

The baseline was implemented by substituting $V_{T^{t-1}}(\{T_i^t, T_j^t\} \to T_m^t)$ in our algorithm with $V_{(Dist)T^{t-1}}(\{T_i^t, T_j^t\} \to T_m^t)$.

6.2 Effectiveness of Constraint Model

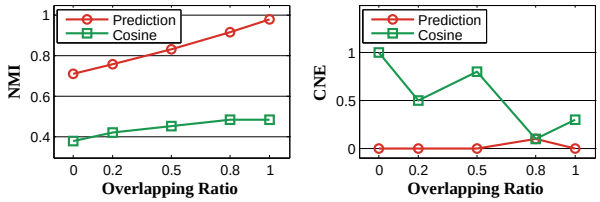
The major goal of this experiment is to evaluate: 1) the effectiveness of the two similarity measures for constraint tree initialization and check which one is better; 2) the clustering quality of the tree built by our constraint model.

6.2.1 Experimental settings

In this experiment, we used the 20 newsgroups dataset. This dataset has a ground-truth hierarchy with two levels of clusters (7 clusters at the first level and 20 clusters at the second level). We removed the clusters with only one child and got a hierarchy with 4 and 17 clusters at the first and second levels, respectively. In each trial, we randomly sampled 2,000 documents to form a ground-truth labeled tree. We treated it as the tree at $t-1$. We then sampled 2,000 documents again and mapped them to the ground-truth labeled tree, which is the initial constraint tree in our model. The second dataset of 2,000 documents can be sampled such that it has a specific percentage of overlap with the first dataset. In our experiment, we considered 5 overlapping ratios that vary from 0 to 1. For each given overlapping ratio, we sampled the two datasets 5 times and the results of each ratio were computed by averaging the results of these 5 trials. The Bayesian rose tree parameter γ was set at 0.1, and $\alpha^{(i)} = 0.01 (i = 1, \ldots, |\mathcal{V}|)$ (Eq. (4)).

6.2.2 Criteria

We evaluated the tree clustering quality by two criteria: Normalized Mutual Information (**NMI**) and Cluster Number Error (**CNE**). **NMI** is the most commonly used metric to measure the clustering quality [22], but may fail in certain instances, especially when the cluster number difference is large. For example, the ground-truth contains 50 clusters, each of which consists of 20 data samples. If the algorithm builds 1,000 clusters, each of which only contain 1 data sample, then the NMI value is 0.75. To solve this problem, we introduced CNE, which measures the cluster number difference between the generated tree and the ground-truth tree at each level. The larger the CNE value, the worse the tree clustering quality. In our experiments, we evaluated the clustering quality by averaging



(a) Overlapping ratio vs. NMI. (b) Overlapping ratio vs. CNE.

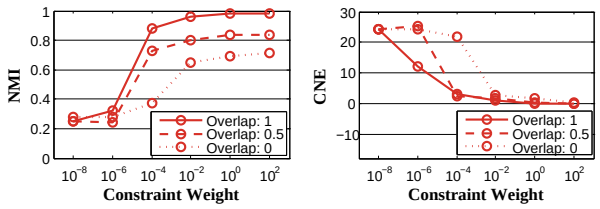
Figure 7: Clustering quality of the constraint trees initialized by different measures with different overlapping ratios.

the NMI/CNE values at different levels. A clustering result with a larger NMI value and smaller CNE value has better quality.

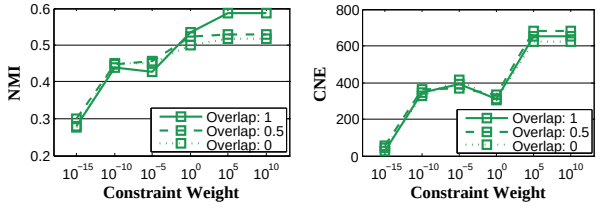
6.2.3 Results

First, we evaluated which similarity measure was better for mapping documents in constraint tree initialization. We used *Cosine* to represent the mapping method with the cosine similarity (Eq. (11)), and *Prediction* to represent the mapping method with the prediction measure (Eq. (12)). Fig. 7 shows how the quality of the initial constraint tree changed with different data overlapping ratios and different measures. The *Prediction* measure outperformed the *Cosine* measure on both criteria. For example, even when the overlapping ratio was 0, the NMI value was 0.7. We speculate that this may be due to the fact that the probability-based prediction measure is more consistent with the tree building probability in our model.

Second, we compared our algorithm with the baseline method to demonstrate its effectiveness in building high-quality tree clustering results. In each experiment, we also compared the results with the overlapping ratios 0, 0.5, and 1. Fig. 8 shows how the tree clustering quality changed with different constraint weight λ (Eq. (9)) for both algorithms with different overlapping ratios. As illustrated by the results, our algorithm is much more effective than the tree distance-based baseline algorithm. When the data overlapping ratio is 1, we can reconstruct the ground-truth labeled tree. Even if the overlapping ratio is 0, our algorithm still maintains a larger NMI (0.7) and smaller CNE (near 0), while the baseline has a smaller NMI (0.5) and much larger CNE (650). In our algorithm, both NMI and CNE become better as the constraint weight increases. In the baseline, the NMI becomes better with the increase of the constraint weight, while the CNE gets worse with the increase of the constraint weight.



(a) Our algorithm, NMI (b) Our algorithm, CNE



(c) Baseline, NMI (d) Baseline, CNE

Figure 8: Tree clustering quality comparison.

6.3 Tree Likelihood vs. Smoothness

In this experiment, we aimed to demonstrate that our algorithm well preserved both fitness and smoothness.

6.3.1 Experimental settings

In this experiment, we used New York Times news articles (from Jan 2006 to Jun 2007)[2]. This dataset contains 12,798 articles on art, style, travel, business, and sports. We grouped the data into nine segments, each of which contained 2 months of articles. We randomly sampled 1,000 documents from each time segment. To eliminate the randomness caused by sampling, we sampled the data 5 times and ran the experiment 5 times. The results were computed by averaging the results of the 5 trials. Bayesian rose tree parameters γ and $\alpha^{(i)}$ were set at 0.03 and 0.0005.

6.3.2 Criteria

In our experiment, we used likelihood to measure the fitness of a tree. We also introduced three metrics to measure the smoothness between adjacent trees.

Tree Distance Smoothness (S_{Dist}): This metric is defined based on the smoothness cost of the baseline algorithm. It measures the tree structure difference by aggregating the tree distance difference between two correlated leaf pairs of the adjacent trees: $S_{Dist} = \frac{1}{\lambda}\log(p_{Dist}(T^t|T^{t-1}))$, where $p_{Dist}(T^t|T^{t-1})$ is defined in Eq. (18).

Tree Order Smoothness (S_{Order}): This metric is defined based on the smoothness cost of our algorithm. It is the negative value of the violated triples/fans compared with the previous tree: $S_{Order} = \frac{1}{\lambda}\log(p(T^t|T^{t-1}))$, where $p(T^t|T^{t-1})$ is defined in Eq. (8).

Robinson-Foulds Smoothness (S_{RF}): This metric is based on the widely used Robinson-Foulds distance metric for phylogenetic trees [20].

$$S_{RF} = -[d_{RF}(T^t, T^{t-1}(\mathcal{D}^t)) + d_{RF}(T^{t-1}, T^t(\mathcal{D}^{t-1}))]/2, \qquad (20)$$

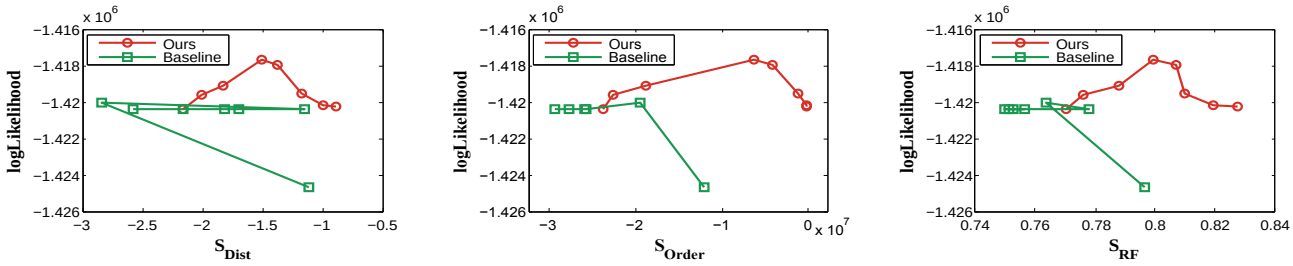
where $T^{t-1}(\mathcal{D}^t)$ represents the constraint tree built on the given tree T^{t-1} and data $\mathcal{D}^t$. Then S_{RF} is defined as the average distance of $(T^t, T^{t-1}(\mathcal{D}^t))$ and $(T^{t-1}, T^t(\mathcal{D}^{t-1}))$. We implemented an improved version of Robinson-Foulds distance given in [14].

6.3.3 Results

Two 5-depth trees were built by the baseline and our algorithm, respectively. Fig.9 shows how the smoothness scores changed with likelihood. We did a grid search of eight constraint weights for both the baseline and our algorithm. The constraint weights for the baseline and our algorithm were $\{1e^{-25}, 1e^{-20}, ..., 1e^{10}\}$ and $\{3e^{-6}, 1e^{-5}, ..., 3e^{-3}, 1e^{-2}\}$. The larger the constraint weight, the more emphasis was put on the smoothness factor. In each of the figures, we connected the points in the order of increasing constraint weights. Based on an analysis of the results, we draw the following conclusions.

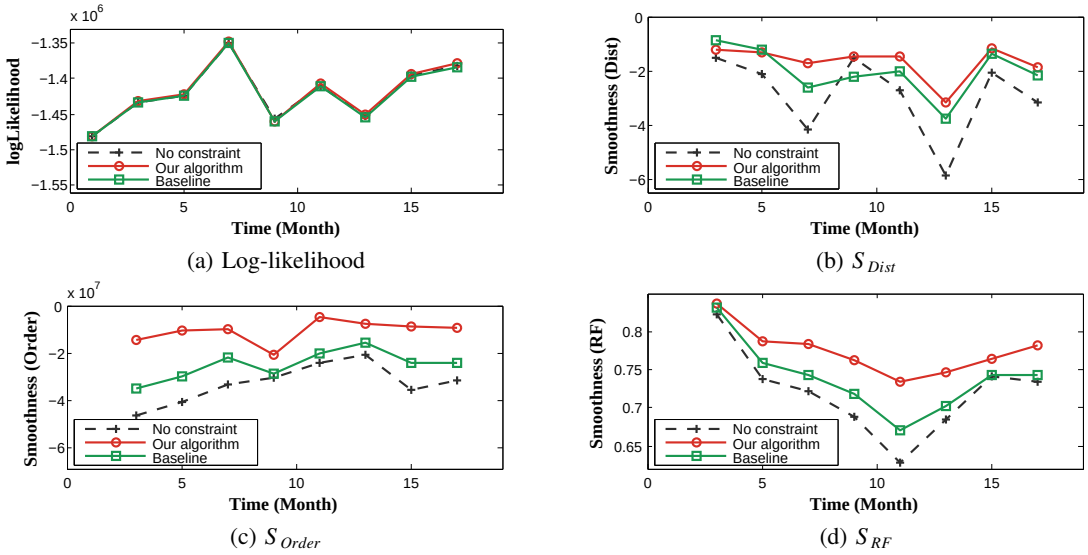
First, our method can generate a much smoother structure than the baseline while maintaining a larger likelihood. Besides having very good performance under the metric of S_{Order}, our algorithm also achieved a comparable performance under the metrics of S_{Dist} and S_{RF}. This demonstrates that triples and fans contain all the hierarchical information of a multi-branch tree and thus the cost function based on them is very effective at preserving both smoothness and fitness. The baseline algorithm achieved a reasonable performance under the metric of S_{Dist}. However, it failed to get a good result under the metrics of S_{Order} and S_{RF}. This is due to the fact that tree distance constraints do not consider the hierarchical information (e.g., parent-child relationships) of a tree.

[2]http://nytimes.com



(a) S_{Dist} vs. log-likelihood (b) S_{Order} vs. log-likelihood (c) S_{RF} vs. log-likelihood

Figure 9: Comparison of smoothness and likelihood with different constraint weights.



(a) Log-likelihood (b) S_{Dist}

(c) S_{Order} (d) S_{RF}

Figure 10: Comparison of smoothness and likelihood at different time points.

Second, the smoothness of our algorithm increases consistently with the increase of the constraint weight, while the likelihood increases at first and then decreases. This indicates that incorporating a certain amount of historical information actually helps to increase fitness. This is because the highly reliable triples/fans are kept in the current tree and they can help the greedy algorithm find a better solution. However the baseline does not exhibit a similar pattern. Even if the constraint weight increases, the smoothness between trees is not guaranteed. This is because the baseline does not well consider multi-branch structures and does not handle conflicting constraints.

Next, we found that our algorithm can well preserve both smoothness and likelihood at each time point. The result was the average values of the eight trails with different constraint weights. As shown in Fig. 10(b), Fig. 10(c), and Fig. 10(d), our algorithm outperformed the baseline on smoothness at almost every time point. Furthermore, the baseline also worked better at preserving smoothness than the method with no constraint (denoted as "No constraint" in Fig. 10), which is equivalent to performing BRT at each time point. As for the likelihood, our algorithm and the baseline were as good as the one with no constraint (Fig. 10(a)). This demonstrates again that our algorithm can preserve smoothness between trees without sacrificing the likelihood of each tree.

6.4 Efficiency

In this section, we first demonstrate that our algorithm is more efficient than the baseline. Then we compare the running time of our algorithm with different constraint weights. Experimental results show that *K*NN-EvoBRT and SpillTree-EvoBRT is as efficient as *K*NN-BRT and SpillTree-BRT.

6.4.1 *Experimental settings*

We randomly sampled 100,000 documents from the New York Times corpus, and evaluated the efficiency of our model based on *K*NN-BRT and SpillTree-BRT. We classified the data into two groups. The first was used for constructing the constraint trees and the second was used for building a new tree based on the constraint tree. The vocabulary size used in this experiment was 665,261.

6.4.2 *Results*

As shown in Fig. 11(a), our algorithm outperformed the baseline in terms of efficiency (based on *K*NN-BRT). Its performance was also comparable to *K*NN-BRT. Fig. 11(b) and Fig. 11(c) demonstrate the running time of *K*NN-EvoBRT and SpillTree-EvoBRT, respectively. Here, we also compare the implementations with different constraint weights. The results demonstrate again that the performance of our algorithm is comparable to that of the two approximation algorithms of BRT. In this example, when the constraint weight was small (i.e., 1e-10), our algorithm was similar to the one with no constraint in terms of efficiency. When the constraint weight was large, our algorithm was even faster for the corpus with a larger number of documents. After checking the results with the faster running time, we found their constraint trees were quite balanced. A balanced constraint tree leads to a balanced tree structure. Typically, BRT and its variations can build a balanced tree much faster. Due

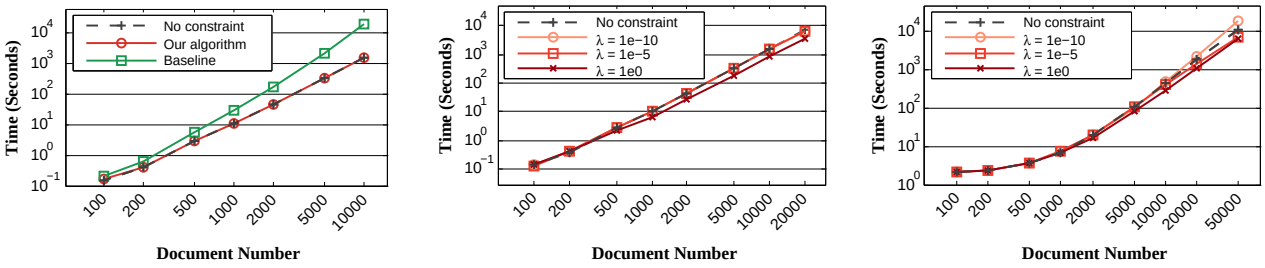


Figure 11: Efficiency comparison: (a) comparison of different methods (KNN); (b) comparison of KNN-EvoBRT ($\lambda = 1e-10, 1e-5, 1e0$) with KNN-BRT (No constraint); (c) comparison of SpillTree-EvoBRT (($\lambda = 1e-10, 1e-5, 1e0$)) with SpillTree-BRT.

to the complexity of our algorithm, it can also handle larger scale datasets in a reasonable amount of time.

7. CONCLUSIONS

In this paper, we present an evolutionary multi-branch hierarchical clustering algorithm, EvoBRT, to automatically learn dynamic tree structures over time. We leverage a Bayesian online filtering framework to formulate our evolutionary clustering problem. To build multi-branch trees, we adopt the state-of-the-art multi-branch tree clustering method, Bayesian rose trees. To preserve tree smoothness over time, we use the conditional prior over tree structures to keep the information from previous trees. Particularly, we introduce the concepts of triples and fans, which can uniquely represent a multi-branch tree. To compute the tree structure differences efficiently, we define a constraint tree from triples and fans, as well as the corresponding operations to make it consistent over time. Our experiments show that our algorithm outperforms the traditional evolutionary hierarchical clustering algorithm both in tree clustering quality and construction efficiency. The complexity analysis demonstrates that our algorithm can be applied to large-scale datasets.

Acknowledgements

The authors would like to thank S. Lin for proofreading, as well as H. Wei and C. Blundell for their insightful discussions.

8. REFERENCES

[1] A. Ahmed, Q. Ho, J. Eisenstein, E. P. Xing, A. J. Smola, and C. H. Teo. Unified analysis of streaming news. In *WWW*, pages 267–276, 2011.

[2] A. Ahmed and E. P. Xing. Dynamic non-parametric mixture models and the recurrent chinese restaurant process: with applications to evolutionary clustering. In *SDM*, pages 219–230, 2008.

[3] A. Ahmed and E. P. Xing. Timeline: A dynamic hierarchical dirichlet process model for recovering birth/death and evolution of topics in text stream. In *UAI*, 2010.

[4] K. Bade and A. Nürnberger. Creating a cluster hierarchy under constraints of a partially known hierarchy. In *SDM*, pages 13–24, 2008.

[5] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *KDD*, 2004.

[6] D. M. Blei and J. D. Lafferty. Dynamic topic models. In *ICML*, pages 113–120, 2006.

[7] C. Blundell, Y. W. Teh, and K. Heller. Discovering non-binary hierarchical structures with Bayesian rose trees. In K. Mengersen, C. P. Robert, and M. Titterington, editors, *Mixture Estimation and Applications*. John Wiley & Sons, 2011.

[8] C. Blundell, Y. W. Teh, and K. A. Heller. Bayesian rose trees. In *UAI*, 2010.

[9] D. Chakrabarti, R. Kumar, and A. Tomkins. Evolutionary clustering. In *KDD*, pages 554–560, 2006.

[10] W. Cui, S. Liu, L. Tan, C. Shi, Y. Song, Z. Gao, H. Qu, and X. Tong. Textflow: Towards better understanding of evolving topics in text. *IEEE TVCG*, 17(12):2412–2421, 2011.

[11] I. Davidson and S. S. Ravi. Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results. *Data Mining and Knowledge Discovery*, 18(2):257–282, 2009.

[12] Z. Gao, Y. Song, S. Liu, H. Wang, H. Wei, Y. Chen, and W. Cui. Tracking and connecting topics via incremental hierarchical dirichlet processes. In *ICDM*, pages 1056–1061, 2011.

[13] N. Kumar, K. Kummamuru, and D. Paranjpe. Semi-supervised clustering with metric learning using relative comparisons. In *ICDM*, pages 693–696, 2005.

[14] Y. Lin, V. Rajan, and B. M. E. Moret. A metric for phylogenetic trees based on matching. *IEEE/ACM TCBB*, 9(4):1014–1022, July 2012.

[15] E. Liu, Z. Zhang, and W. Wang. Clustering with relative constraints. In *KDD*, pages 947–955, 2011.

[16] X. Liu, Y. Song, S. Liu, and H. Wang. Automatic taxonomy construction from keywords. In *KDD*, pages 1433–1441, 2012.

[17] R. E. Madsen, D. Kauchak, and C. Elkan. Modeling word burstiness using the Dirichlet distribution. In *ICML*, pages 545–552, 2005.

[18] S. Miyamoto and A. Terami. Constrained agglomerative hierarchical clustering algorithms with penalties. In *FUZZ-IEEE*, pages 422–427, 2011.

[19] M. P. Ng and N. C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69(1-2):19–31, August 1996.

[20] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

[21] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS*, 2003.

[22] A. Strehl and J. Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, March 2003.

[23] X. Wang, C. Zhai, X. Hu, and R. Sproat. Mining correlated bursty topic patterns from coordinated text streams. In *KDD*, pages 784–793, New York, NY, USA, 2007. ACM.

[24] X. Wang, K. Zhang, X. M. Jin, and D. Shen. Mining common topics from multiple asynchronous text streams. In *WSDM*, pages 192–201, 2009.

[25] T. B. Xu, Z. F. Zhang, P. S. Yu, and B. Long. Dirichlet process based evolutionary clustering. In *ICDM*, pages 648–657, 2008.

[26] T. B. Xu, Z. F. Zhang, P. S. Yu, and B. Long. Evolutionary clustering by hierarchical Dirichlet process with hidden Markov state. In *ICDM*, pages 658–667, 2008.

[27] D. Zhang, C. Zhai, J. Han, A. N. Srivastava, and N. C. Oza. Topic modeling for olap on multidimensional text databases: topic cube and its applications. *Statistical Analysis and Data Mining*, 2(5–6):378–395, 2009.

[28] J. Zhang, Y. Song, C. Zhang, and S. Liu. Evolutionary hierarchical Dirichlet processes for multiple correlated time-varying corpora. In *KDD*, pages 1079–1088, 2010.

[29] H. Zhao and Z. Qi. Hierarchical agglomerative clustering with ordering constraints. In *WKDD*, pages 195–199, 2010.

[30] L. Zheng and T. Li. Semi-supervised hierarchical clustering. In *ICDM*, pages 982–991, 2011.