International Journal of Database Theory and Application
Vol.7, No.3 (2014), pp.107-120
http://dx.doi.org/10.14257/ijdta.2014.7.3.11

Mining Sequential Trees in a Tree Sequence Database

Yijun Bei[1], Zhen Lin[2]*, Qiyao Wang[3] and Erteng Liu[1]

[1]College of Software Technology, Zhejiang University, China
[2] College of Computer Science & Technology, Zhejiang University;
Department of Computer Science, University of Illinois at Urbana - Champaign, USA
[3] State Key Laboratory of Network and Switching Technology,
Beijing University of Posts and Telecommunications, China
nblin@zju.edu.cn

Abstract

Tree structures are used extensively in domains such as XML data management, web log analysis, biological computing, and so on. In this paper we introduce the problem of mining frequent sequential trees in a large tree sequence database. We present a framework for mining frequent sequential trees in a so-called tree sequence database. Basically, this framework employs a transformation-based approach which converts the sequential tree mining problem into a traditional sequence mining problem. Our approach firstly mines frequent trees in the tree sequence database. Secondly, we perform a database transformation by means of tree-containment computation to generate a sequence database. Thirdly, after the transformation, frequent sequence patterns can be mined in the newly created sequence database using a conventional sequence mining technique. Finally we perform an inverted transformation process on the output of sequence mining to obtain sequential tree patterns. Experimental results on synthetic datasets show that the proposed framework is both effective and efficient in finding frequent sequential trees in a large tree sequence database.

Keywords: *tree mining, sequence mining, tree sequence mining, tree pattern*

1. Introduction

In recent years, semi-structured data have become ubiquitous with the rapid upsoaring in both number and scale of semi-structured data applications such as XML database systems, business transactions, XML middleware systems, and so on. As the size of the data increases, efficient retrieval of semi-structured data becomes critical for such applications. For example, XML queries can be expedited by deploying a cache of frequent queries. This issue can be addressed by mining frequent XML query patterns, which are in the form of trees in nature, to the XML database. As another example, accesses to a website, where web pages are often stored in a tree organization, can be improved considerably using a cache if the most frequently accessed subtrees are found. Therefore, mining frequent tree patterns is an interesting topic in many performance-critical database applications. The frequent patterns that our mining technique could find include, but are not limited to, tree patterns such as XML query patterns, web access patterns, protein patterns and so on.

There are a few previous works in the literature proposed to discover tree patterns. However these works do not consider the temporal feature or other sequential features. In our work, we will illustrate the effectiveness of exploiting the sequential features of the trees.

* Corresponding Author

Actually, our method could be interpreted as a solution to the problem of sequential semi-structured data mining. The main contributions of our work are as follows:

(1) We bring forward the problem of mining frequent sequential patterns in a tree sequence database and define a problem statement of frequent sequential tree mining.

(2) We present a framework for solving the problem of sequential tree mining in a tree database. The framework consists of four component steps: tree mining, database transforming, sequence mining, and sequence transforming.

(3) We propose an approach to transform a tree sequence database into a traditional sequence database. Firstly, we employ an existing sequence mining algorithm in order to mine frequent trees in the original tree sequence database. Secondly, we transform the sequential tree mining problem into a traditional sequence mining problem using the frequent trees mined in the first step. We note that our method is not dependent on a specific sequence mining algorithm being used. So the framework can flexibly work with other sequence mining algorithms at the expense of trivial efforts.

(4) For the final output, we give a method of transforming mined sequences into non-redundant sequential trees.

(5) We implement a prototype tree-mining system. The experiments show that our proposed method could generate good mining results on synthetic datasets.

There may be several types of tree structures, such as rooted ordered tree, rooted unordered tree, free tree. Without loss of generality, we focus on the sequential mining problem of root ordered tree in our work. Simple modification to the framework would allow us to mine other types of tree-structures.

The rest of the paper is organized as follows. In Section 2, we discuss previous work related to sequence mining and tree mining. In Section 3, we prepare basic definitions and define the problem of mining frequent sequential trees in a tree sequence database. We present a framework for sequential tree mining in Section 4. Section 5 gives the results of experiments and we conclude our work in Section 6.

2. Related work

Many efficient mining approaches have been developed [1-7] to discover frequent sequences. Agrawal [1] proposed sequential pattern mining first and developed a generalized and refined algorithm GSP [2] later. SPADE presented by Zaki [5] can avoid many repeated database scans and mine sequences in only three database scans. SPADE makes use of combinatorial properties to decompose the original problem into smaller sub-problems that can be independently solved in main-memory with lattice search techniques. During search, a vertical id-list database format is adopted and frequent sequences are enumerated via simple temporal joins on id-lists. Han et al. introduced FreeSpan [3] and PrefixSpan [4] to the problem of sequence mining. PrefixSpan employs a horizontal format database representation and utilizes the pattern-growth paradigm: adopting a prefix growth and database projection framework and an optimization pseudo-projection is used when projecting databases. MCPRISM in [21] uses prime encoding that effectively mines min-closed sequences.

For the problem of finding tree-like patterns, many efficient tree mining approaches have been developed [8-14]. Basically, there are two main steps for generating frequent trees in a database (forest). First of all, a systematic way should be conceived for generating non-redundant candidate trees whose support is to be computed. Secondly, an efficient way is required to compute the support of each candidate tree and determine whether a tree is frequent. Anyhow, they adopted a straight-forward generate-and-test strategy. Asai presented a rooted ordered and a rooted unordered tree mining approach in [12] and [13] respectively. Zaki gave ordered and unordered embedded tree mining algorithms in [8, 9]. Chi *et al.*, brought forward algorithms for mining rooted unordered and free trees in [10, 11]. In [15], Bei *et al.*, proposed a tree mining algorithm named BUXMiner for finding rooted unordered trees. Liu *et al.*, [20] proposed an algorithm to find all sequential patterns in the updated database.

XML query can be considered as a tree pattern. Frequent query access patterns can be made use of accelerating retrieval of query results [16-19]. Yang *et al.*, presented a series of apriori-style algorithms called XQPMiner, XQPMinerTID, and FastXMiner to discover frequent rooted query pattern subtrees from a set of XML queries in [16, 17]. Ling *et al.*, in [19] take into account the temporal features of user queries to discover association rules. They clustered XML queries according to their semantics first and then mined association rules between clusters for caching. We also introduced FLS method to represent XML documents and a framework of clustering algorithm using frequent sequences in [22].

3. Problem Statement

In this section, we define some basic concepts that will be used in the remainder of the paper and state the problem about sequential tree mining.

3.1. Rooted Ordered Tree

A **rooted ordered tree** is an undirected, connected, acyclic graph $T = (V, E, \sum, L, r, \leq)$ consisting of a *vertex* set V, an *edge* set E, an *alphabet* $\sum$ for vertex labels and a *labeling* function L. The rooted ordered tree T with a distinguished node r called root satisfying the following properties. If vertex u is on the path from the root to vertex v then u is the ancestor of v and v is the descendent of u. If in addition u and v are adjacent, then u is the parent of v and v is the child of u. The mapping function L: $V \rightarrow \sum$ assigns a label $L(v)$ to each node $v \in V$. In an ordered tree, the children of each vertex are ordered. The binary relation $\leq \subseteq V^2$ represents a *sibling* relation between two children of the same parent. We denote $v \leq w$ if vertex v and vertex w have the same parent u and v is an elder brother of w.

Given two rooted ordered trees T and S on an alphabet $\sum$, we say that S is a subtree of T iff there exists a one-to-one mapping φ: $V_S \rightarrow V_t$, such that satisfies the following conditions:

(1) φ preserves the labels, *i.e.*, $L(v) = L(\varphi(v))$, $\forall v \in V_S$.

(2) φ preserves the parent relation, *i.e.*, $(u,v) \in E_S$ iff $(\varphi(u), \varphi(v)) \in E_T$.

(3) φ preserves the sibling relation, *i.e.*, $v \leq w$ for $\forall v \in V_S$ and $\forall w \in V_S$ iff $\varphi(u) \leq \varphi(v)$ for $\forall \varphi(v) \in V_T$ and $\varphi(w) \in V_T$.

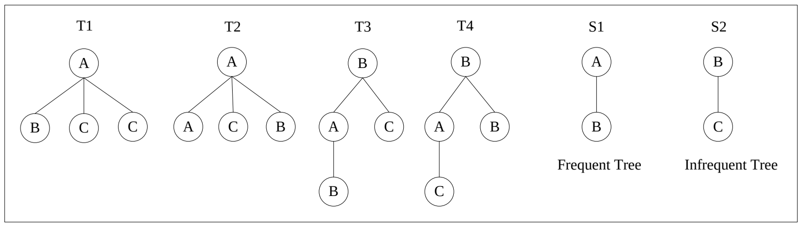


Figure 1. Frequent Tree and Infrequent Tree

If a tree S is a **subtree** of a tree T, we denote it as $S \subseteq T$. We also can say that T contains S or S occurs in T. In Figure 1, we show that S_1 is a subtree of trees T_1,T_2 and T_3 while S_2 is a subtree of the only tree T_3.

3.2. Frequent Tree

Let D denotes all the trees in a forest and d_T be an indicator variable with $d_T(S) = 1$ if tree S is a subtree of T and $d_T(S) = 0$ if tree S is not a subtree of T. The frequency (support) of tree S in the forest can be defined as $\sigma(S) = \sum_{T \in D} d_T(S)$, *i.e.*, the number of trees in D that contains tree S. A tree is frequent in a forest if its frequency is no less than a user-specified *minimum* **support.** The tree is considered as a **frequent tree** in a forest. In Figure 1, the frequency of S_1 is 3 and the frequency of S_2 is 1 for the forest with trees T_1, T_2, T_3 and T_4. Suppose the minimum support is 3, then by definition, tree S_1 is frequent while S_2 is not in the forest.

3.3. Tree Sequence Database

A **tree sequence database** (or a **sequential database of trees**) is a database with each record containing a set of trees and trees in each record are in a certain sequential order (The order can be temporal order, spatial order, *etc*.). A tree in a tree sequence database can be denoted as (*rid*, *tid*), where *rid* is the id of record containing the tree and *tid* is the tree id in the record. Figure 2 shows a sequential database of trees and trees in each record are ordered. There are 3 records in the database and each record contains 5 sequential trees. The first tree at the top-left corner can be represented as (R1, 1). Let f_R be an indicator variable with $f_R(S) = 1$ if there exists a tree $T \in R$ with $d_T(S) = 1$ and $f_R(S) = 0$ if there is no tree T belonging to R that contains S. The frequency (support) of tree S in a tree sequence database can be defined as $\sigma(S) = \sum_{R \in D} f_R(S)$. A tree is frequent in a tree sequence database D if its frequency is no less than a user-specified *minimum* **support**.

3.4. Sequential Tree Mining Problem

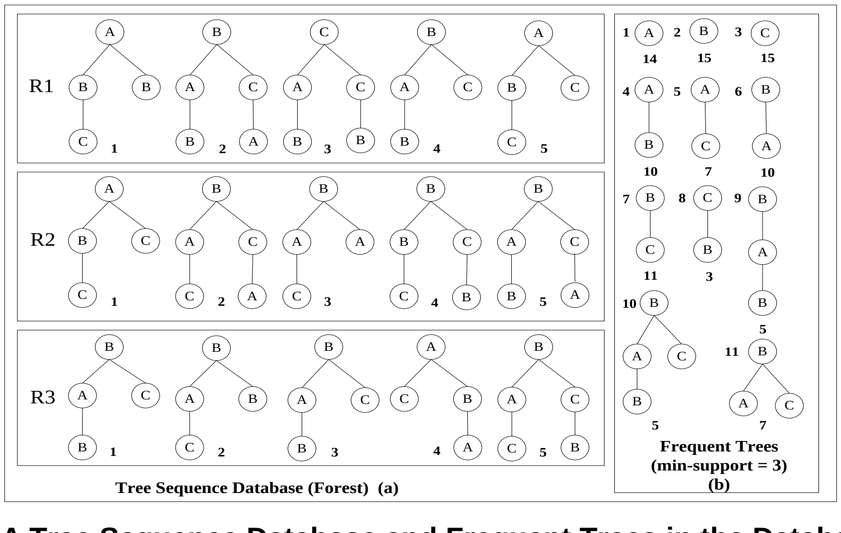


Figure 2. A Tree Sequence Database and Frequent Trees in the Database (a) R1 Denotes the Record id of the Database; Label under the Tree is the Tree id in Current Record. (b) Label at the Left of Tree is the Frequent Tree id and the Value under the Tree is the Number of Trees Containing Current Tree in the Database

Let $T = \{t_1, t_2, \ldots, t_m\}$ be a set of all subtrees in D. A **tree sequence** is an ordered list of subtrees. Let P and Q be two tree sets, $P \leq Q$ is denoted as all trees $p \in P$, there exists a tree $q \in Q$ that contains p. We can say that tree sets Q contains sets P. A tree sequence S can be denoted by $<s_1, s_2, \ldots, s_n>$, where s_j is a set of trees, *i.e.*, $s_j \subseteq T$.

One tree sequence $L = \{l_{i1}, l_{i2}, \ldots, l_{im}\}$ is a **subsequence** of another $S = \{s_1, s_2, \ldots, s_n\}$ if $\forall 1 \leq j \leq m\text{-}1$, $i_j < i_{j+1}$ and $\forall 1 \leq j \leq m$, $\exists 1 \leq k \leq n$ such that $l_{ij} \leq s_k$ (s_k contains l_{ij}). In this case, tree sequence S contains tree sequence L. The tree sequence database is a set of records (*rid*, *S*), where *rid* is the identifier of current sequential trees and *S* is a tree sequence. The **support (frequency)** of a tree sequence S in a tree sequence database is the number of records in the database that contain S, denoted as $\text{freq}_D(S)$. A tree sequence is frequent in a tree sequence database if its support is more than or equal to a user-specified *minimum* support. We consider it as a frequent tree sequence in the tree sequence database. As show in Figure 2, suppose the user-given *minimum* support is 3, and then tree sequence <(1,4),(1,5)> (Number 1, 4 and 5 respectively represent labeled id of trees in the set of frequent trees in Figure 2 (b)) is frequent in the database since records *R1*, *R2* and *R3* all contain the tree sequence.

4. Framework of Sequential Tree Mining

In this section, we present a framework for frequent tree sequence mining in the tree sequence database. In Figure 3, we give a presentation of the whole framework. The framework of tree sequence mining consists of the following major parts: (1) **Tree Mining**, *i.e.*, given a user specified minimum support and an input tree sequence database, a set of frequent trees are mined from the database; (2) **Database Transforming**, with the previous generated frequent trees we transform the tree sequence database into a general sequence database; (3) **Sequence Mining**, frequent subsequences are mined from the new generated sequence database; (4) **Sequence Transforming**, frequent subsequences generated in step (3) are transformed into non-redundant frequent tree sequences which is our final goal.

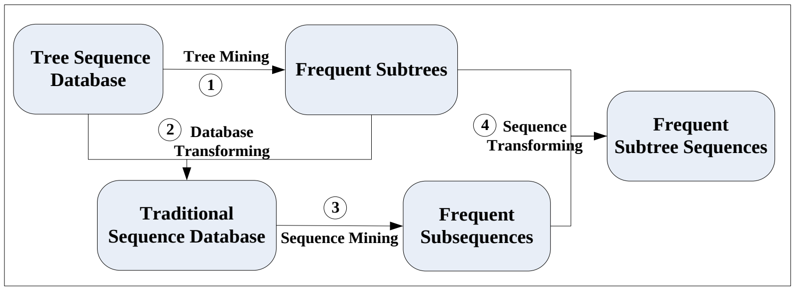


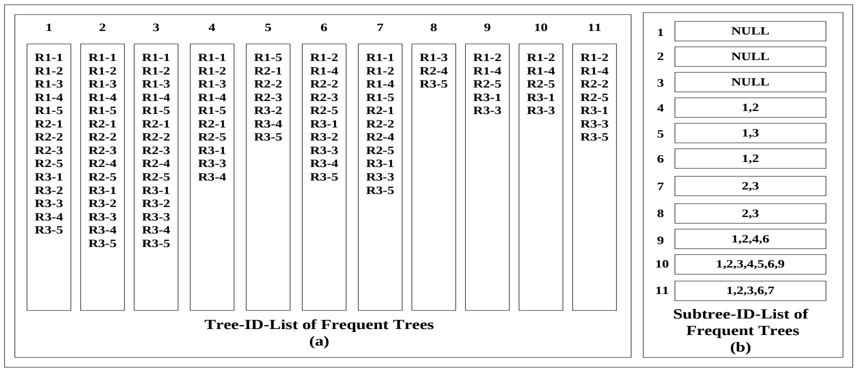
Figure 3. A Framework of Tree Sequence Mining in the Tree Sequence Database

4.1. Tree Mining

Many efficient mining approaches have been developed for finding tree-like patterns [8-13]. In our paper, since we focus mainly on rooted ordered tree mining problem, we choose the tree mining algorithm **FreqT** from Asai. However, other tree mining approaches can also be used for analysis of other tree-structures.

We do not use FreqT algorithm to mine frequent trees in a tree sequence database straight. On the contrary, we make several modifications to the FreqT algorithm. First of all, since we mine frequent trees in a tree sequence database while not in a tree database, the computation of support of trees should be modified. A tree is frequent in a tree database while it may not be frequent when considering the database as a tree sequence database. Secondly, when mining frequent trees, we label each frequent tree with a unique id. In Figure 2 (b), there are 11 frequent trees and we label trees from 1 to 11 respectively. For each frequent tree we use a *tree-id-list* to record all the trees in database that contain current frequent tree. In this way we can obtain the *tree-id-list* during mining process and avoid a time-costly tree-containing computation. When computing the support of each candidate tree, trees in database are enumerated in database order (first by *rid* order then by *tid* order). Therefore, constructing results of *tree-id-list*s are also automatically ordered in database order without any other disposal. In Figure 4 (a), we list the *tree-id-list* of every frequent tree. We use Ri-k to denote a tree with Ri to be *rid* and *k* to be *tid*. For example, the No. 10 frequent tree is a subtree of trees {R1-2, R1-4, R2-5, R3-1, R3-3}. For every frequent tree we also use a *subtree-id-list* to record all the frequent trees contained by current frequent tree. Frequent trees recorded in *subtree-id-list* are order by the frequent tree id. Figure 4 (b) shows the results of *subtree-id-list* for each frequent tree in Figure 2. For example, the No. 1 frequent tree does not contain any other frequent tree while No. 9 tree contains frequent trees {1, 2, 4, 6}. Generation of *subtree-id-list* will be used during process of sequence transforming. A detail of its function will be discussed when talking about sequence transforming.

In case that the database is large we cannot store all *tree-id-list* and *subtree-id-list* in memory, we can store all id-lists in disk and load them when needed. All the *tree-id-list*s are stored first by the frequent tree id order and then by the database order. All the *subtree-id-list*s are stored first by the frequent tree id order and for each *subtree-id-list* it is stored also by frequent tree id order.



1	2	3	4	5	6	7	8	9	10	11
R1-1	R1-1	R1-1	R1-1	R1-5	R1-2	R1-1	R1-3	R1-2	R1-2	R1-2
R1-2	R1-2	R1-2	R1-2	R2-1	R1-4	R1-2	R2-4	R1-4	R1-4	R1-4
R1-3	R1-3	R1-3	R1-3	R2-2	R2-2	R1-4	R3-5	R2-5	R2-5	R2-2
R1-4	R1-4	R1-4	R1-4	R2-3	R2-3	R1-5		R3-1	R3-1	R2-5
R1-5	R1-5	R1-5	R1-5	R3-2	R2-5	R2-1		R3-3	R3-3	R3-1
R2-1	R2-1	R2-1	R2-1	R3-4	R3-1	R2-2				R3-3
R2-2	R2-2	R2-2	R2-5	R3-5	R3-2	R2-4				R3-5
R2-3	R2-3	R2-3	R3-1		R3-3	R2-5				
R2-5	R2-4	R2-4	R3-3		R3-4	R3-1				
R3-1	R2-5	R2-5	R3-4		R3-5	R3-3				
R3-2	R3-1	R3-1				R3-5				
R3-3	R3-2	R3-2								
R3-4	R3-3	R3-3								
R3-5	R3-4	R3-4								
	R3-5	R3-5								

Tree-ID-List of Frequent Trees
(a)

1	NULL
2	NULL
3	NULL
4	1,2
5	1,3
6	1,2
7	2,3
8	2,3
9	1,2,4,6
10	1,2,3,4,5,6,9
11	1,2,3,6,7

Subtree-ID-List of Frequent Trees
(b)

Figure 4. Tree-ID-List and Subtree-ID-List of Frequent Trees

4.2. Database Transforming

During this process, we can transform the tree sequence database into a traditional sequence database. An approach for database transforming is given in Algorithm 1. The input of the algorithm is *tree-id-list* of all frequent trees and the output is a new generated sequence database. Each id in tree-id-list can be represented as a three-tuple format (*rid*, *tid*, *fid*)[1], where *rid* is the record id of tree, *tid* is the tree id in the record and *fid* is the frequent tree id. Since tree ids in each *tree-id-list* are ordered according to the tree order in database (first by *rid* order then by *tid* order). First of all, we can sort ids in all *tree-id-list*s first by database order then by frequent tree id order with a merge sort. A data structure heap is used for sorting ids. If the amount of ids is too large, we can use an external merge sort approach.

[1] The tree-id-list of a frequent tree is transformed from the 2-tuple format <rid, tid> into 3-tuple format <rid, tid, fid> by appending id of the frequent tree to the original format.

Secondly, we generate the records of new database. The records are sequences and elements in sequences are itemsets. Ids with a same *rid* and *tid* constitute an itemset for a tree. All itemsets with a same *rid* constitute a sequence (record). Itemsets are ordered by *tid* in the same record. By transforming a tree in the tree sequence database can be represented as an itemset which composed by all frequent trees contained by current tree. In Figure 5(a), we show the new sequence database generated from *tree-id-list*s in Figure 4 (a). To be briefly, we only use *fid* set to denote an itemset. Items in itemset {1,2,3,4,7} are all with *rid* = R1,*tid* = 1 and represent tree (R1,1). Itemsets in sequence <{1,2,3,4,7}, {1,2,3,4,6,7,9,10,11}, {1,2,3,4,8}, {1,2,3,4,6,7,9,10,11}, {1,2,3,4,5,7}> are all with *rid* = R1 and represent record R1.

```
GenerateDatabase
Input: TREE_ID_lISTS
Output: DATABASE
result_lists = Φ;
compute merge_lists from TREE_ID_lISTS
//a merge_list in merge_lists with structure (start_list,end_list)
for each merge list ml in merge_lists do
      result_list = Merge(TREE_ID_lISTS,ml);
 result_lists = result_lists ∪ result_list
      if count of list in result_lists is more than 1 then
    DATABASE = GenerateDatabase(result_lists);
      else
            DATABASE = ClusterTreeID(result_lists);
 end if
end for

Merge
Input: TREE_ID_lISTS, merge_list
Output: result_list
Construct a priority Queue TreeIDMergeQueue do
for each list id li in merge_list do
     get a tree id list til from TREE_ID_lISTS according to li
  get next tree id ti from til;
  if  ti exists then
     insert ti into TreeIDMergeQueue
  end if
end for
while TreeIDMergeQueue is not empty do
      pop top tree id tti from TreeIDMergeQueue
   insert tti into result_list
      get next tree id ti from tree_id_list which tti is in;
      if ti exists then
        insert ti into TreeIDMergeQueue
      end if
end while

ClusterTreeID
Input: TREE_ID_LIST
Output: DATABASE
for each tree id ti in tree_id_list do
      if tree id ti and prev tree id pti not with same record id then
      generate a new record with rid equals ti.rid;
      DATABASE = DATABASE ∪ {record};
   else if tree id ti and prev tree id pti not with same tree id then
     generate a new tree with tid equals ti.tid;
     record = record ∪ {tree};
   else
     tree = tree ∪ {ti.fid};
   end if
end for
```

Algorithm 1. Database Transformation Algorithm

4.3. Sequence Mining

For the sake of discovering sequential patterns, many efficient mining approaches have been developed [1-6]. Among these efficient algorithms, we adopt algorithm SPADE to mine sequences from our new generated sequence database as it is a more efficient and easy to implement one. SPADE presented by Zaki [5] can avoid many repeated database scans and mine sequences in only three database scans. SPADE makes use of combinatorial properties to decompose the original problem into smaller sub-problems that can be independently solved in main-memory with lattice search techniques. Further detailed discussion on the SPADE algorithm can be found in [5]. In Figure 5(b) we show the frequent sequences discovered from the generated database in Figure 5(a) by SPADE.

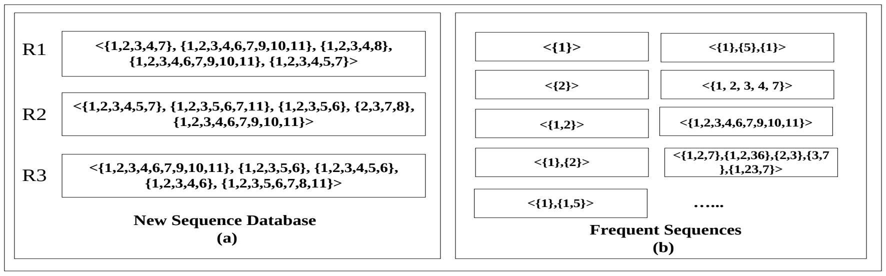


Figure 5. Generated Sequence Database and Frequent Sequences in the Database

4.4. Sequence Transforming

In this section, we give a scheme to change mined sequences into non-redundant tree sequences. Itemset constituting mined sequence is composed of all frequent trees that the tree contains. Since a tree *S* is contained in a *T*, then all subtrees of *S* will also be contained in *T*. Therefore, there may exist some redundant data in mined sequences. For example, in Figure 5 (b), sequence <{1, 2, 3, 4, 7}> with only one itemset is a frequent tree sequence in database. While No. 1 and 2 trees are subtrees of No. 3 tree, and No. 4 tree is a subtree of No. 7 tree. By removing redundant items, a new non-redundant sequence {<4, 7>} is constructed. Thus, a tree sequence <{4, 7}> is a non-redundant frequent tree sequence, with No. 4 and No. 7 trees are two frequent trees respectively.

```
SequenceTransforming
Input: Sequence
Output: NewSequence
NewSequence = <>;
for each itemset in Sequence do
  for each item t in itemset from back to front do
    for each item s before item t do
      if size(s) < size(t) AND contain(t,s) then
        itemset = itemset - {s};
      end if
    end for
    end for
  NewSequence = NewSequence ◊ itemset
end for
```

Algorithm 2. Non-redundant Sequence Transformation Algorithm

To remove redundant items, we need a time-costly tree-containing computation. Fortunately, we have computed the tree-containing relation during frequent tree mining process. We use contain(*t*, *s*) to denote a tree *t* contains a tree *s* and size(*t*) denote the number of nodes in tree *t*. In Algorithm 2, a non-redundant sequence transformation algorithm is given. Items (frequent trees) in each itemset are ordered in frequent tree id order which is increased when a new frequent tree generated. When generating frequent trees, a bread-first enumerate approach is employed. Therefore, frequent trees are ordered by tree size. With the tree size order property, we can have that size of the tree in the same itemset is equal or larger than previous one. We use tree size order property and tree-containing relation to judge the relationship between two frequent trees for sequence transforming.

5. Experimental Results

In this section, we present the results of actual implementation of the tree sequence mining algorithm for a tree sequence database.

5.1. Implementation and Experimental Setup

A prototype system of tree sequence mining algorithm is implemented in Java programming language (Sun JDK1.6). As described in previous sections, the prototype system of the mining algorithm is made up of four parts: frequent tree mining, database transforming, sequence mining and sequence transforming. We compared time consumption of the four parts for various datasets. All experiments were performed on a server (Intel 2.0GHz Pentium dual machine) with 4GB of main memory running operating system Ubuntu 11.10.

5.2. Workload

We used tree generator provided by Zaki[2] to generate a number of datasets with varying sizes. For generating trees, following default values were used for the parameters: the maximum fanout *F*=10, the maximum depth *D*=10, the number of labels *N*=100, the number of nodes in the master tree *M*=10,000. Various synthetic datasets are used for representing varying number of trees: T10K for total number of trees *T*=10,000, T1M for *T*=1,000,000. To form a tree sequence database, every 5 trees was used to comprise a record. Thus there are 2,000 records for T10K dataset and every sequence contains 5 trees. In the following experiments, the default relative minimum support value was set 10%.

5.3. Mining Performance

Table 1. Running Time of Four Parts of Tree Sequence Mining System with Various Datasets

Part / Dataset	TM(ms)	DT(ms)	SM(ms)	ST(ms)	Total(ms)
T10K	1896	2568	3474	613	8551
T20K	2661	4848	4337	489	12335
T40K	4456	8519	8292	379	21646
T80K	8141	16923	16239	404	41707
T100K	10019	20806	21013	424	52262
T200K	19334	40920	36182	438	96874
T400K	38651	82228	76022	413	197314
T800K	78585	151933	156327	476	387321
T1M	95159	209082	232087	399	536727

[2] http://www.cs.rpi.edu/~zaki/software/.

In Table 1 we show running time of four parts of tree sequence mining system with various datasets, where TM stands for Tree Mining, DT stands for Database Transforming, SM stands for Sequence Mining and ST stands for Sequence Transforming. From the table we can see that the running time is mainly consumed by Database Transforming and Sequence Mining. There are many sorting and merging operations during Database Transforming which may cost much I/O. During sequence mining we have to iterate all candidate subsequence. Although we decompose the problem into smaller sub-problems that can be independently solved in main-memory and prune infrequent sequences as early as possible, there are still too many candidates need to be examined. The time consumed by sequence mining is still large.

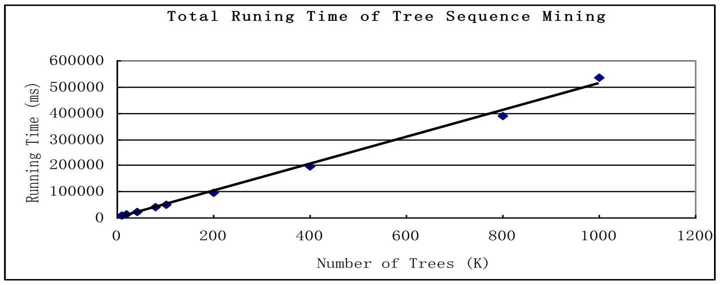


Figure 6. Total Running Time of Tree Sequence Mining

Figure 6 shows total time consumed by frequent tree sequence mining with datasets of varying size from T10K to T1M. From the figure we can see that the time consumed by the mining system is related to the dataset size linearly, which means our approach has good scalability and shows a good performance for sequence tree mining.

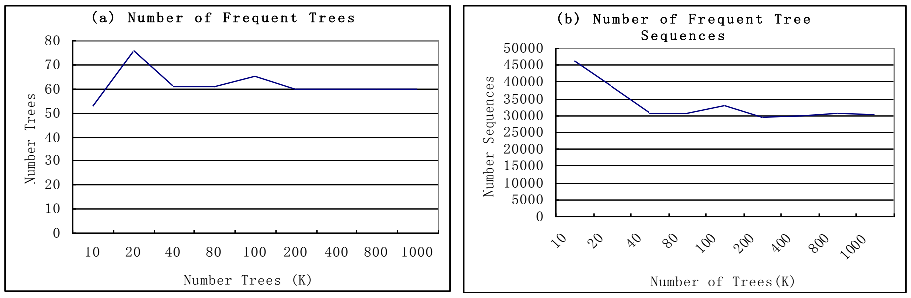


Figure 7. Number of Frequent Trees and Frequent Tree Sequences with Various Datasets

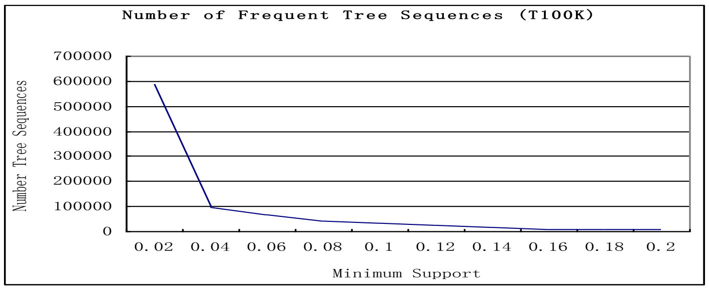


Figure 8. Number of Frequent Tree Sequences for Dataset T100k with Various Supports

Figure 7 (a) shows the number of frequent trees mined from datasets of varying size, while Figure 7 (b) shows the number of frequent sequence trees. Since we adopt relative support when mining tree sequences, the number of frequent trees for one dataset is almost equal to others. The result of the number of frequent tree sequence behaves likewise. Therefore, the running time of Sequences Transforming in Table 1 varies little. Figure 8 presents the number of frequent tree sequences for dataset T100k with various minimum supports from 0.02 to 0.20, which shows that the number of mined tree sequences decreases rapidly when support is on increase.

6. Conclusions

This paper brought forward the problem about mining frequent sequential patterns in a tree sequence database, and presented a framework for sequential tree mining in a tree sequence database. We decomposed the process of the framework into four steps: tree mining, database transforming, sequence mining and sequence transforming. We presented our method for each of these steps. We conducted experiments on synthetic datasets, and found that the running time of our approach is linear to the size of the tree sequence database. The experimental results also indicate that our system has good scalability. For future work, we plan to apply our approach to applications such as cache for XML query, and web server cache, *etc*.

Acknowledgements

The work was supported in part by National Natural Science Foundation of China [grant number 60903038], China Scholarship Council [grant number 201306320037], National Key Technology R&D Program [grant number 2013BAH01B06], Smart Industry Talent Base Project of Ningbo.

References

[1] R. Agrawal and R. Srikant, Mining sequential patterns, Proceedings of the 11th International Conference on Data Engineering **(1995)**, March 6-10; Taipei, China.
[2] R. Srikant and R. Agrawal, Mining Sequential Patterns: Generalizations and performance improvements, Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology **(1996)**, March 25-29; Avignon, France.
[3] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal and M.-C. Hsu, PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, Proceedings of the 17th International Conference on Data Engineering **(2001)**, April 2-6; Heidelberg, Germany.
[4] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal and M.-C. Hsu, FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining, Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining **(2000)**, August 20 – 23; Boston, MA, USA.
[5] M. J. Zaki, SPADE: An Efficient Algorithm for Mining Frequent Sequences, In Machine Learning Journal, special issue on Unsupervised Learning, vol. 42, no. 1/2, pp. 31-60 **(2001)**.
[6] M. N. Garofalakis, R. Rastogi and K. Shim, SPIRIT: Sequential pattern mining with regular expression constraints, Proceedings of the 25th International Conference on Very Large Data Bases, pp. 223-234 **(1999)**, September 7-10; Edinburgh, Scotland, UK.
[7] B. Mallick, D. Garg and P. S. Grover, Incremental mining of sequential patterns: Progress and challenges, Intelligent Data Analysis, vol. 17, no. 3, pp. 507-530 **(2013)**.
[8] M. J. Zaki, Efficiently Mining Frequent Trees in a Forest, Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining (2002), July 23-25; Edmonton, AB, Canada.
[9] M. J. Zaki, Efficiently Mining Frequent Embedded Unordered Trees, Fundamenta Informaticae, vol. 66, no. 1-2, pp. 33-52 **(2004)**.
[10] Y. Chi, Y. Yang and R. R. Muntz, Indexing and Mining Free Trees, Proceedings of the 3rd IEEE International Conference on Data Mining **(2003)**, November 19-22; Melbourne, Florida, USA.

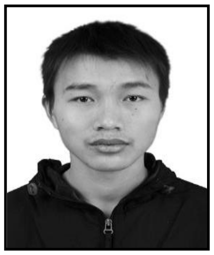
[11] Y. Chi, Y. Yang and R. R. Muntz, HybridTreeMiner: An Efficient Algorithm for Mining Frequent Rooted Trees and Free Trees Using Canonical Forms, Proceedings of the 16th International Conference on Scientific and Statistical Database Management **(2004)**, June 21-23; Santorini Island, Greece.
[12] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Satamoto and S. Arikawa, Efficient Substructure Discovery from Large Semi-structured Data, Proceedings of the 2nd SIAM International Conference on Data Mining **(2002)**, April 11-13; Arlington, VA, USA.
[13] T. Asai, H. Arimura, T. Uno and S. Nakano, Discovering Frequent Substructures in Large Unordered Trees, Proceedings of the 6th International Conference on Discovery Science **(2003)**, October 17-19; Sapporo, Japan.
[14] C. T. Jiang, F. Coenen and M. Zito, A survey of frequent subgraph mining algorithms, The Knowledge Engineering Review, vol. 28, no. 1, pp. 75-105 **(2013)**.
[15] Y. Bei, G. Chen, L Shou, X. Li and J. Dong, Bottom-up discovery of frequent rooted unordered subtrees, Information Sciences, vol. 179, no. 1-2, pp. 70-88 **(2009)**.
[16] L. Yang, M.L. Lee and W. Hsu, Efficient mining of XML query patterns for caching, Proceedings of the 29th International Conference on Very Large Data Bases, pp. 69–80 **(2003)**, September 9-12; Berlin, Germany.
[17] L. Yang, M.L. Lee, W. Hsu, D. Huang and L. Wong, Efficient mining of frequent XML query patterns with repeating-siblings, Information and Software Technology, vol. 50, no. 5, pp. 375-389 **(2008)**.
[18] M. Mazuran, E. Quintarelli and L. Tanca, Data Mining for XML Query-Answering Support, IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 8, pp. 1393-1407 **(2012)**.
[19] L. Chen, S. S. Bhowmick and L.-T. Chia, Mining Positive and Negative Association Rules from XML Query Patterns for Caching, Proceedings of the 10th international conference on Database Systems for Advanced Applications, pp.736-747 **(2005)**, April 17-20; Beijing, China.
[20] J. X. Liu, S. T. Yan, Y. Y. Wang and J. D. Ren, Incremental Mining Algorithm of Sequential Patterns Based on Sequence Tree, Advances in Intelligent Systems, vol. 138, pp. 61-67 **(2012)**.
[21] K. Ravi, H. S. Srinivasan and S. K. Krishnan, Comment spam detection by sequence mining, Proceedings of the 5th ACM international conference on Web search and data mining, pp. 183-192 **(2012)**, February 8-12; Seattle, Washington, USA.
[22] Y. J. Bei, X. L. Zheng and Z. Lin, Evaluating the Similarity of XML Documents Based on Frequent Label Sequences, International Journal of Advancements in Computing Technology (IJACT), vol. 4, no. 13, pp. 26-34 **(2012)**.

Authors

Yijun Bei, received the BS and PhD degrees in the College of Computer Science & Technology of Zhejiang University in 2003 and 2008 respectively. He worked as a lecturer in the College of Software Technology, Zhejiang University. His research interests include databases data mining, XML data mining and social network analysis.

Zhen Lin, is currently a Ph.D. candidate at Zhejiang University. Also, he becomes a joint Ph.D. student at University of Illinois at Urbana - Champaign since 2013. He received his Bachelor degree from Northeastern University, and Master degree from Zhejiang University. He joined the E-Business Technology Institute (ETI) of the University of Hong Kong in 2009. His research interests include social computing, data mining, and community detection.

Qiyao Wang, is currently a Ph.D. candidate at Beijing University of Posts and Telecommunications. Also, she becomes a joint Ph.D. student at University of Illinois at Urbana - Champaign since 2014. She received his Bachelor degree from Beijing University of Posts and Telecommunications. Her research interests include social computing and information diffusion.



Erteng Liu, is currently a lecturer at the Zhejiang University since 2012. He received his Bachelor degree from Guizhou University, and Master degree from Zhejiang University. His research interests include data analysis, social computing, and software engineering.