A Structure Preserving Flat Data Format Representation for Tree-Structured Data

Fedja Hadzic

DEBII, Curtin University, Perth, Australia
f.hadzic@curtin.edu.au

Abstract. Mining of semi-structured data such as XML is a popular research topic due to many useful applications. The initial work focused mainly on values associated with tags, while most of recent developments focus on discovering association rules among tree structured data objects to preserve the structural information. Other data mining techniques have had limited use in tree-structured data analysis as they were mainly designed to process flat data format with no need to capture the structural properties of data objects. In this paper, we present a novel structure-preserving way for representing tree-structured document instances as records in a standard flat data structure to enable applicability of a wider range of data analysis techniques. The experiments using synthetic and real world data demonstrate the effectiveness of the proposed approach.

Keywords: XML mining, tree mining, decision tree learning from XML data

1 Introduction

Semi-structured documents such as XML possess a hierarchical document structure, where an element may contain further embedded elements, and each element can be attached with a number of attributes. It is therefore frequently modeled using a rooted ordered labeled tree. Many frequent subtree mining algorithms have been developed that driven by different application aims mine different subtree types [1-7]. Even though they are well scalable for large datasets, if the instances in the tree database are characterized by complex tree structures the approach will fail due to enormous number of candidate subtrees that need to be enumerated, as was analyzed mathematically in [5]. For an overview of the current state-of-the-art in the field of tree-structured data mining, please refer to [3, 8, 9].

There has been limited work in classification methods in tree-structured data. Some initial work was mainly based on tree-structured association rules and queries [10, 11], while very recently work presented in [12] employs a mathematical programming method to directly mine discriminative patterns as numerical feature. The XRules approach [13] is a rule based classification system based on the frequent subtree patterns discovered by the TreeMiner algorithm [6], and hence the structural

information is taken into account during model learning. However, frequent subtree patterns can be very large in number, many of which may not be useful for the classification task at hand, and one often needs to filter out many of irrelevant/uninteresting patterns using a variety of statistical and heuristic measures. Several methods using neural networks and kernel methods for processing tree-structured data have been proposed [14], where the approaches are typically tailored towards the use of a particular machine learning method. In this work, an alternative approach is taken with the focus on the data conversion process not tailored to any specific method, yet enabling the application of available methods for data in tabular form in general.

A unique and effective way of representing tree-structured data into a flat data structure format is proposed. The structural information is preserved in form of additional attributes, but the results can differ to those obtained using frequent subtree mining based approaches. A discussion is therefore provided regarding the scenarios in which the proposed approach can be particularly useful and can extract knowledge when other frequent subtree based approaches would fail due to inherent complexity. This is supported with an example case in which all closed/maximal subtrees can be obtained using a closed/maximal itemset mining method on the proposed representation, which as experimentally shown can significantly improve the performance. In the remainder of the experiments, the focus is on the application of a decision tree learning method, to evaluate its usefulness when applied on a synthetic and real-world tree-database converted into the proposed format. The preliminary results indicate the great potential in enabling the application of well established and theoretically/practically proven data analysis techniques that otherwise could not process tree-structured data directly. In addition, the use of quality and interestingness measures developed for relational data can be explored in tree-structured data context.

2 Background of the Problem

A graph consists of a set of nodes (or vertices) that are connected by edges. Each edge has two nodes associated with it. A path is defined as a finite sequence of edges. A rooted tree has its top-most node defined as the root that has no incoming edges. In a tree there is a single unique path between any two nodes. A node u is said to be a parent of node v , if there is a directed edge from u to v . Node v is then said to be a child of node u . Nodes with the same parent are called siblings. The *fan-out/degree* of a node is the number of children of that node. The *level/depth* of a node is the length of the path from root to that node. The *Height* of a tree is the greatest level of its nodes. A *rooted ordered labelled tree* can be denoted as $T = (v_0, V, L, E)$, where (1) $v_0 \in V$ is the root vertex; (2) V is the set of vertices or nodes; (3) L is a labelling function that assigns a label $L(v)$ to every vertex $v \in V$; (4) $E = \{(v_1, v_2) \mid v_1, v_2 \in V \text{ AND } v_1 \neq v_2\}$ is the set of edges in the tree, and (4) for each internal node the children are ordered from left to right. The problem of *frequent subtree mining* can be generally stated as: given a database of trees Tdb and minimum support threshold (σ), find all subtrees that occur at least σ times in Tdb . Most commonly mined subtrees are induced and

embedded [9]. Given a tree $S = (vs_0, V_S, L_S, E_S)$ and tree $T = (vt_0, V_T, L_T, E_T)$, S is an *ordered induced subtree* of T iff (1) $V_S \subseteq V_T$; (2) $L_S \subseteq L_T$, and $L_S(v) = L_T(v)$; and (3) $E_S \subseteq E_T$, and (4) the left to right ordering of sibling nodes in the original tree is preserved. Given a tree $S = (vs_0, V_S, L_S, E_S)$ and tree $T = (vt_0, V_T, L_T, E_T)$, S is an *ordered embedded subtree* of T iff (1) $V_S \subseteq V_T$; (2) $L_S \subseteq L_T$, and $L_S(v) = L_T(v)$; (3) if $(v_1, v_2) \in E_S$ then $\text{parent}(v_2) = v_1$ in S and v_1 is ancestor of v_2 in T , and (4) the left to right ordering of sibling nodes in the original tree is preserved. To reduce the number of candidate subtrees that need to be enumerated, some work has shifted toward the mining of maximal and closed subtrees [7]. A closed subtree is a subtree for which none of its proper supertrees has the same support, while for a maximal subtrees, no supertrees exist that are frequent. No information is lost since the complete set of frequent subtrees can be obtained from closed subtrees (including support information) and maximal subtrees. For formal definitions of frequent subtree mining related concepts and overview of the tree mining field we refer the interested reader to [3, 8, 9].

The common way of representing trees is first discussed, to lay the necessary ground for explaining our tree-structured data to flat data format conversion process. The work described in this paper will be utilizing the pre-order (depth-first) string encoding (φ) as described in [6]. In the remainder of the paper we will refer to pre-order string encoding simply as string encoding. A *pre-order traversal* can be defined as follows: If ordered tree T consists only of a root node r , then r is the pre-order traversal of T . Otherwise let $T_1, T_2, \dots, T_n$ be the subtrees occurring at r from left to right in T . The pre-order traversal begins by visiting r and then traversing all the remaining subtrees in pre-order starting from T_1 and finishing with T_n . The pre-order string encoding [6] can be generated by adding vertex labels in a pre-order traversal of a tree $T = (v_0, V, L, E)$, and appending a backtrack symbol (for example ‘-1’, ‘-1’ $\notin L$) whenever we backtrack from a child node to its parent node. In Fig. 1 we show an example tree database Tdb consisting of 6 tree instances (or transactions) ($T_0, \dots, T_5$) and the pre-order string encoding is shown below each tree (i.e. $\varphi(T_1) = \text{‘k r -1 m -1’}$).

Fig. 1. Example of a tree-structured database consisting of 6 transactions

3 Proposed Tree-Structured to Flat Data Method

This section describes our technique to convert the string like presentation commonly used by frequent subtree mining algorithms [6] into a flat data structure format (henceforth referred simply as table) so that both structural and attribute-value

information is preserved. The first row of a (relational) table consists of attribute names, which in a tree database are scattered through independent tree instances (transactions). One way to overcome this problem is to first assume a structure according to which all the instances/transactions are organized. Each of the transactions in a tree-structured document should be a valid subtree of this assumed structure, which we refer to as the *model tree* (*MT*). This *MT* will become the first row of the table, and while it does not contain the attribute names, it contains the most general structure where every instance from the tree database can be matched to. The *MT* needs to ensure that when the labels of a particular transaction from the tree database are processed, they are placed in the correct column, corresponding to the position in the *MT* where this label was matched to. Hence, the labels (attribute names) of this *MT* will correspond to pre-order positions of the nodes of the *MT* and sequential position of the backtrack ('-1') symbols from its string encoding.

The process of extracting a *MT* from a tree database consists of traversing the tree database and expanding the current *MT* as necessary so that every tree instance can be matched against *MT*. Let the tree database consisting of n transactions be denoted as $Tdb = \{tid_0, tid_1, \dots, tid_{n-1}\}$, and let the string encoding of the tree instance at transaction tid_i be denoted as $\phi(tid_i)$. Further, let $|\phi(tid_i)|$ denote the number of elements in $\phi(tid_i)$, and $\phi(tid_i)_k$ ($k = \{0, 1, \dots, |\phi(tid_i)|-1\}$) denote the k_{th} element (a label or a backtrack '-1') of $\phi(tid_i)$. The same notation for the string encoding of the (current) *MT* is used, i.e. $\phi(MT)$. However, rather than storing the actual labels in $\phi(MT)$, 'x' is always stored to represent a node in general. The process of extracting the *MT* from *Tdb* can be explained by the pseudo code below.

Model Tree Extraction from a Tree database *Tdb*

Input: *Tdb*

Output: *MT*

```

inputNodeLevel = 0 // current level of  $\phi(tid)_k$ 
MTNodeLevel = 0 // current level of  $\phi(T(h_{max}, d_{max}))_k$ 
 $\phi(MT) = \phi(tid_0)$  // set default MT (use 'x' instead of labels)
for  $i = 1$  to  $n - 1$  //  $n = |Tdb|$ 
    for each  $\phi(tid)_k$  in  $\phi(tid_i)$ 
        for each  $p = 0$  to  $(|\phi(MT)|-1)$ 
            if  $\phi(tid)_k = -1$  then inputNodeLevel-- else inputNodeLevel++
            if  $\phi(MT)_p = 'b_i'$  then MTNodeLevel-- else MTNodeLevel++
            if inputNodeLevel  $\neq$  uniformNodeLevel
                if  $\phi(tid)_k = -1$  then
                    while inputNodeLevel  $\neq$  MTNodeLevel
                        p++
                        if  $\phi(MT)_p = -1$  then MTNodeLevel-- else MTNodeLevel++
                    endwhile
                else
                    while inputNodeLevel  $\neq$  MTNodeLevel
                        append 'x' at position  $p+1$  in  $\phi(MT)$ 
                        k++
                        p++
                        if  $\phi(tid)_k = -1$  then inputNodeLevel-- else inputNodeLevel++
                    endwhile
            endif
        endfor
    endfor
endfor
return MT

```

To illustrate the complete conversion process using MT please refer back to Fig. 1. Using the string encoding format representation [6], the tree database Tdb from Fig. 1 would be represented as is shown in Table 1, where the left column corresponds to the transaction identifiers, and the right column is the string encoding of each subtree.

Table 1. Representation of Tdb (Fig. 1) in string encoding representation

Tdb	
0	k r n -1 p -1 -1 m -1
1	k r -1 m -1
2	b a e -1 c -1 -1 c d -1 e -1 -1
3	b c d -1 -1
4	b a c -1 -1 c d -1 e -1 -1
5	k r p -1 -1

In this example the MT is reflected in the structure of T_2 in Fig. 1 and it becomes first row in Table 2 to reflect the attribute names as explained before. We use the string encoding to represent this uniform structure and since the order of the nodes (and backtracks ('-1')) is important we label the nodes and backtracks sequentially according to their occurrence in the string encoding. For nodes (labels in the string encoding), x_i is used as the attribute name, where i corresponds to the pre-order position of the node in the tree, while for backtracks, b_j is used as the attribute name, where j corresponds to the backtrack number in the string encoding. Hence, from our running example in Fig. 1 and Table 1, $\phi(MT) = 'x_0 x_1 x_2 b_0 x_3 b_1 b_2 x_4 x_5 b_3 x_6 b_4 b_5'$. To fill in the remaining rows every transaction from Tdb is scanned and when a label is encountered it is placed to the matching column (i.e. under the matching node (x_i) in the uniform structure), and when a backtrack ('-1') is encountered, a value '1' (or 'y') is placed to the matching column (i.e. matching backtrack (b_j) in MT). The remaining entries are assigned values of '0' (or 'no', indicating non existence). The flat data format of Tdb from Table 1 (and Fig. 1) is illustrated in Table 2.

Table 2. Flat representation of Tdb in Fig.1 and Table 1

x_0	x_1	x_2	b_0	x_3	b_1	b_2	x_4	x_5	b_3	x_6	b_4	b_5
k	r	n	1	p	1	1	m	0	0	0	0	1
k	r	0	0	0	0	1	m	0	0	0	0	1
b	a	e	1	c	1	1	c	d	1	e	1	1
b	c	d	1	0	0	1	0	0	0	0	0	0
b	a	c	1	0	0	1	c	d	1	e	1	1
k	r	p	1	0	0	1	0	0	0	0	0	0

Implications of the proposed approach. The model tree MT governs what is to be considered as a valid instance of a particular tree characteristic, and the exact positions of the node within the MT are taken into account. Hence if two subtrees have the same labels and are structurally the same, but their nodes have been matched against different nodes in MT (i.e. they occur at different positions) they would be considered as different characteristic instances. Consider the subtree with encoding 'b c d' from the Tdb in Fig. 1. Within the current tree mining framework this subtree

would be considered to occur in 3 transactions T_2 , T_3 and T_4 , while using the proposed approach the occurrence of ‘b c d’ in T_3 would be considered different than in T_2 and T_4 , because nodes ‘c’ and ‘d’ occur at different (pre-order) positions, and hence the instance is represented differently in the table as can be seen in Table 2. This illustrates the key difference, and it is caused by the fact that not all the instances of a tree database may follow the same order or will have all the elements of the document input structure (e.g. XML schema) available. However, if all the instances in a tree database always follow the same structure and node layout as the input document structure, then there would be no such difference in the method. For example the proposed method would be more applicable in cases when there is an XML Schema available where each XML instance conforms to this schema and contains all the elements in the same order. This is demonstrated with an experiment at the end of this section. However, it is first worth mentioning that, in some applications, it may be the case that the different occurrence of a pattern in the document tree indicates that it is used in different context.

Fig. 2. An XML fragment from an instance of process log data

As an example, please consider Fig. 2 which shows a fragment of a process instance from a process log dataset obtained from <http://prom.win.tue.nl/tools/prom>. Let us say, that the user wants to find out whether the action ‘CheckStatus’ was performed at a particular stage in the workflow. As can be seen in Fig. 2, this action occurs multiple times within a process instance, and it needs to be distinguished based on its occurrence within the workflow model. In cases when the users are investigating the conformance of business process instances to a business process model, it may be important to know when an action or sets of actions occur in a phase of the workflow other than expected by the model, and hence the differing characteristic of the proposed method may be desired. Another difficulty we encountered in real world applications of frequent subtree mining, is related to the complexity issues caused by a number of attributes that are likely present in every instance or transaction. To provide an illustrative example, consider an application of credit risk assessment in banks for providing loans to small to medium enterprises (SMEs). Qualitative data on loan applications can usually be found from text databases while the quantitative information is usually stored in relational format. We have proposed our industry partner (bank) the use of XML to capture domain specific terms and effectively organize the available quantitative and qualitative information. The XML template was produced based on a small number of textual document

instances provided [15]. A simplified portion of the XML template is shown in Fig. 3 to show some of the information stored and the possible sets of values.

```
<XML>
<creditapplication>
<creditriskindicators>
<personalinfo>
  <industryrisk>very low,low,moderate,high,very high</industryrisk>
  <creditinformation>
    <principal>up to IDR 50 million,more than IDR 50 million up to IDR 100
    million</principal>
    <duration>up to 1 year,more than 1 year to 2 years,more than 2 years to 3 years,more
    than 3 years to 4 years,more than 4 years to 5 years</duration>
  </creditinformation>
</personalinfo>
...
<customerprofile>
  ...
  <integrity>high, medium, low</integrity>
  <capacity>
    <salesperyear>below IDR 300 million, above IDR 300 million to IDR 2.5 billion,above
    IDR 2.5 billion to IDR 50 billion</salesperyear>
    <eatpermonth>less than 0.2 of sales, between 0.2 to 0.4 of sales,more than 0.4 of
    sales</eatpermonth>
  </capacity>
</customerprofile>
<financialreport>
<currentratio>below standard, above standard</currentratio>
....
<salesgrowthratio>below standard, above standard</currentratio>
</financialreport>
</creditriskindicators>
<creditassessment>accept,accept with considerations,reject</creditassessment>
</creditapplication>
</XML>
```

Fig. 3. XML Template (part of) for credit risk assessment of SMEs

There are a total of 53 tags (attributes) which store additional and more specific information (eg, industry, sub-industry, etc.). At this stage we could not obtain sufficient amount of document instances from the bank as the rejected applications were not kept. We have conducted several interviews with the loan officers from the bank in order to obtain some domain knowledge in form of rules that reflect realistic credit parameters and the corresponding assessment decision. This resulted in obtaining 63 rules that can be applied to reflect a realistic assessment decision for the given criteria. A total of 100,000 XML instances were created based upon the pre-defined XML template, with random values for credit parameters. The 63 rules provided by domain experts were then incorporated on top of those instances to modify the class value realistically. We have removed each instance whose criteria values could not be found in a rule, which reduced the dataset to 53,081 instances. These rules were general with respect to the financial ratios, indicating that a ratio is above or below a standard for a particular industry/sub-industry. However, real ratios were used for the corresponding industry/sub-industry, giving a total of 41 possible variations where a general rule can apply. While on average a single rule would have modified 800 instances, the rule itself will not be present in 800 instances because specific ratios apply. Hence, low support thresholds would need to be used, but this has shown to be impractical because of the large amount of subtree patterns generated. This is caused by 13 nodes that are present in each instance and are mainly

there to contextualize the information (e.g. creditriskindicators, customerprofile). As these nodes are present in patterns for any given support, combinatorial complexity problems occur at lower support thresholds. We ran the IMB3-miner [4] algorithm for mining ordered induced subtree, but because of the just mentioned issues, enumerating all induced subtrees that occurred in less than half of the database was infeasible. Directly mining for the class discriminative factors would be preferred.

In this dataset the information in every instance is organized according to the pre-defined XML template and hence the key difference of mining the converted tree database in flat format may not occur. Due to the subset exclusion properties of closed and maximal pattern one would expect that the number of closed/maximal itemsets should be the same as the number of closed/maximal subtrees for a given support. To confirm this we have compared the results of the CMTreeMiner [7] algorithm that extracts closed and maximal ordered induced subtrees together with the results of Charm [16] and GenMax [17], that extract closed and maximal itemsets, respectively.

Fig. 4. Time performance of closed/maximal subtree and itemset mining

The results from Charm and GenMax required some post-processing, as nodes are not ordered and every backtrack ('-1') is present in the maximal/closed itemset. However, since the patterns contain attribute names, any node can be matched directly to the model tree (MT), and the right structural organization of the nodes worked out. Hence, we are sequentially listing the values of each matched node in MT , and whenever a level of the previously listed node (L_{prev}) is $\leq$ level of the currently listed node (L_{cur}), we append $((L_{cur} - L_{prev}) + 1)$ number of the backtracks after the currently listed node. Since the MT itself is ordered according to the pre-order traversal, this results in pre-order string encodings of the subtrees, giving the same results as CMTreeMiner. In Fig. 4, we show the time performance comparison between CMTreeMiner and the sum of time taken by GenMax and Charm algorithms for varying support thresholds. When support threshold was set to 1% CMTreeMiner was taking over two hours and the run was terminated. As expected, enumerating frequent closed/maximal itemsets is faster than enumerating frequent closed/maximal subtrees, because in the latter the structural validity needs to be ensured during the candidate generation phase. In fact further performance gain could be achieved by removing all the backtrack attributes from the sets of items, as they are worked out based upon the occurrence of nodes in MT . Hence, in scenarios where the schema is available, and all nodes are unique and the instances are organized in the same order,

one can mine for maximal/closed itemsets from the converted flat representation to obtain patterns when approaches for mining closed/maximal subtrees would fail due to inherent structural complexity. Little post-processing is required but as experimentally shown the performance gain can be large.

4 Experimental Evaluation

The purpose of the experiments provided in this section is to demonstrate how using the proposed tree database transformation one can discover interesting knowledge using other data mining techniques than association rule mining. We focus on the application of the C4.5 decision tree learning algorithm [18] and Weka software [19] is used.

Synthetic Data. In this experiment we consider several synthetic datasets specifically designed for the problem of classifying each tree instances according to its structural properties. Hence, the attribute values are not what can be used to classify each instance correctly, but rather it is the structural characteristics of each instance that determines its correct class. Hence, the used datasets and the classification problem mainly serves the purpose of evaluating the capability of the proposed representation approach to capture the structural properties of each tree instance and take them into account for classification.

The first dataset was created as follows. Each tree instance was accompanied with a class attribute, which indicates the maximum height and degree of that particular tree instance (transaction). For example, if the maximum height is 2, and the maximum degree is 3, then the class label will be 23_val. For the first test, the aim was to generate a database that contains every possible instance of a tree for a given maximum height of 2 and degree 4. We have created one tree with height 2, where all nodes have degree 4 and have the same label ('2'), from which we have enumerated all possible induced subtrees. This will ensure that every instance of the class is present and there were a total of 9 classes (note that '00_val' is used for a tree consisting of a single node).

The model tree (*MT*) is therefore a tree with height 2 where all nodes have degree 4, as all class instances can be matched against this tree. Each attribute name is flagged on the node/edge of the *MT* tree structure displayed in Fig. 5. Since we have chosen the same label for all nodes, the node attributes will have the same predictive capability as the backtrack attributes. This would usually not be the case if the variety of labels are used, as it is through the backtrack attributes that the structural properties are preserved. Fig. 6 shows the learned C4.5 decision tree for this dataset. Two tests were performed so that in the second test (right side of Fig. 6) all of the attributes of node type (' x_i ') were removed from learning, as in reality the node labels will be different and should not have predictive capability in this classification task. Model building took 0.03 seconds, and the 10-fold cross-validation test option was used and both decision trees had accuracy of 99.23%. Please note that in this test the class value distribution was very uneven, because of the way that the dataset was generated, as for higher degree the number of possible unique instances of a class is much larger. There were 695 instances of class 24_val (i.e. max height = 2 and max degree = 4), 71

of 23_val, 9 of 22_val, while the remaining 6 classes had only one representative instance. We have repeated the test when the number of classes was made the same, and there was an increase in accuracy to 99.85%, but the decision tree was much larger, i.e. size 53 and 27 leaves. By comparing Figures 5 and 6 one can verify the extracted rules.

Fig. 5. Illustrating the attribute names for synthetic database

Fig. 6. C4.5 decision trees for synthetic dataset

The second experiment performed for this classification task has used the TreeGenerator software [6] to generate synthetic database of 50,000 records. The maximum height and degree were set to 5, giving us a total of 26 unique class values. Hence in this case each node was assigned a random label and it is not guaranteed that every instance of a particular class will be generated. The C4.5 decision tree learned from this data had 99.748% accuracy, it consisted of 1278 nodes and 1153 leaves and it took 34.57 seconds to obtain. This complex tree could be partly because the value for option n (number of items) was set to 1000. By analyzing the decision tree we notice that while some backtracks (b_i) were used for instance splitting during decision tree construction, many constraints on labels for nodes (x_i) occur at the leaf nodes. This indicates that particular labels happen to be present/absent at a particular node for many instances of a particular class. The experiment was repeated on a dataset generated with n option in TreeGenerator [6] set to 1 (i.e. all nodes share

1 label). The resulting decision tree reduced to only have 299 nodes and 150 leaves, with the classification accuracy of 99.786%, and the time taken was 24.88 seconds. These results demonstrate that the structural and attribute/value information is preserved and taken into account for the classification task at hand.

Web Log Data. This experiment set utilizes the publically available real-world server log data (US1924, US2430, US304) previously used in [13] to evaluate their XRules method. These datasets have a variety of tree structures, and each dataset contains a number of very deep and wide trees at the end. This kind of property is undesired in the proposed approach as the model tree (*MT*) from the database will be fairly large and many of the nodes and backtracks will be there because of a small number of tree instances that were out of the norm with respect to the general structure. Hence we perform two test sets, one in which we process the dataset as originally found (e.g. US1924 in Table 3) and one where we only pick the first 7000 records as these do not contain those rare large instances (e.g. 7KUS1924 in Table 3). The accuracy (10-fold cross-validation) and size of each C4.5 decision tree learned from this data is shown in Table 3. This is a challenging classification problem for the proposed method. The dataset has a variety of small and large trees. The larger trees do not occur as frequently, but they still need to be captured by the *MT*. This makes it much larger than necessary to capture the majority of trees in the database (hence the long running time when compared to 7K variants in Table 3). However, the results are comparable to the results of the XRules structural classifier presented in [13].

Table 3. C4.5 results for the Web log datasets

Datasets	US1924	7KUS1924	US2430	7KUS2430	US304	7KUS304
Accuracy(%)	76.68	84.4	76.75	81.91	83.18	77.29
Size	13222	12079	14882	16405	21460	20813
Leaves #	13195	12060	14862	16394	21413	20781
Time(seconds)	7044.16	53.14	4743.16	45.41	6825.56	37.66

5 Conclusion and Future Work

This paper has presented a unique and effective way of representing tree structured databases such as XML documents into a flat data format for which many more available data mining/analysis methods exist in comparison to available methods for tree-structured data format. The preliminary experimental results have demonstrated that the structural and attribute/value information is preserved in the proposed flat data representation. A standard decision tree learning algorithm performed very well in classifying tree instances according to their structural properties. Regarding, the real world dataset not particularly suitable for the proposed presentation, the decision tree learning algorithm still achieved comparable classification accuracy with previously reported results based on a frequent subtree mining approach. More studies will be performed in future and the necessary extensions for the approach to perform well in a wider range of cases. Generally speaking, enabling effective representation of all information from a tree-structured database in a flat data format, will give the

opportunity to apply and evaluate well established and theoretically proven data analysis techniques that could not process tree-structured data directly.

References

1. Abe, S.K., Kawasoe, S., Asai, T., Arimura, H., Arikawa, S.: Optimized substructure discovery for semistructured data. In: Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery, Helsinki, Finland, pp 1–14 (2002)
2. Tatikonda, S., Parthasarathy, S., Kurc, T.: TRIPS and TIDES: new algorithms for tree mining. In: ACM CIKM'06, Arlington, Virginia, USA (2006)
3. Chi, Y., Nijssen, S., Muntz, R.R., Kok, J.N.: Frequent Subtree Mining - An Overview. *Fundamenta Informaticae*, Special Issue on Graph and Tree Mining, 66:1-2 (2005)
4. Tan, H., Dillon, T.S., Hadzic, F., Feng, L., Chang, E.: IMB3-Miner: Mining Induced/Embedded Subtrees by Constraining the Level of Embedding. In: 10th Pacific-Asia Conf. on Knowledge Discovery and Data Mining, Singapore, pp 450–461 (2006)
5. Tan, H., Hadzic, F., Dillon, T.S., Feng, L., Chang, E.: Tree Model Guided Candidate Generation for Mining Frequent Subtrees from XML. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 2(2) (2008)
6. Zaki, M.J.: Efficiently mining frequent trees in a forest: algorithms and applications. *IEEE Transaction on Knowledge and Data Engineering*, vol. 17(8), pp. 1021-1035 (2005)
7. Chi, Y., Yang, Y., Xia, Y., Muntz, R.R.: CMTreeMiner: Mining both closed and maximal frequent subtrees. In: PAKDD'04, pp.63-73, Sydney, Australia (2004)
8. Tan, H., Hadzic, F., Dillon, T.S., Chang, E.: State of the art of data mining of tree structured information. *Int'l Journal Computer Systems Science and Eng.*, vol. 23, no 2, March (2008)
9. Hadzic, F., Tan, H., Dillon, T. S. 2011. Mining of Data with Complex Structures. *Studies in Computational Intelligence Series*, vol. 333, Springer, Berlin/Heidelberg, Germany.
10. Chen, L., Sourav, S.B., Chia, L-T.: Mining Association Rules from Structural Deltas of Historical XML Documents. In: PAKDD 2004, pp. 452-457, Sydney, Australia, (2004)
11. Braga, D., Campi, A., Ceri, S., Klemettinen, M., Lanzi, P.: Discovering interesting information in XML data with association rules. In: ACM Symposium on Applied Computing, pp. 450-454, Melbourne, Florida (2003)
12. Kim, H., Kim, S., Weninger, T., Han, J., Abdelzaher, T.: NDPMine: Efficiently Mining Discriminative Numerical Features for Pattern-Based Classification. In: Proc. of ECML PKDD 2010, Barcelona, Spain, Sept 20-24 (2010).
13. Zaki, M.J., Aggarwal, C.C.: XRules: An Effective Structural Classifier for XML Data. In: SIGKDD '03, Washington DC, USA (2003)
14. Da San Martino, G., Sperduti, A.: Mining Structured Data. *IEEE Computational Intelligence Magazine*, vol 5(1) (2010)
15. Ikarari, N., Hadzic, F., Dillon, T.S.: Incorporating Qualitative Information for Credit Risk Assessment through Frequent Subtree Mining for XML. *XML Data Mining: Models, Method, and Applications* (ed) A. Tagarelli, IGI Global (2011)
16. Zaki, M. J., Hsiao, C.-J.: CHARM: An Efficient Algorithm for Closed Itemsets Mining. In: 2nd SIAM Int'l Conf. on Data Mining, Arlington, VA, USA, April 11-13 (2002)
17. Gouda, K., Zaki, M.J.: GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets, *Data Mining and Knowledge Discovery*, vol. 11, no. 3, pp.223-242 (2005)
18. Quinlan, J.R.: C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, San Mateo, CA (1993).
19. Holmes, G., Donkin, A., Witten, I.H.: Weka: A machine learning workbench, In: 2nd Australia and New Zealand Intelligent Information Systems Conf, Brisbane, Australia, 1994.