Mining tree patterns with Almost Smallest Supertrees

Jeroen De Knijf *
Department of Information and Computing Science
Universiteit Utrecht
jknijf@cs.uu.nl

Abstract

In this work we describe a new algorithm to mine tree structured data. Our method computes an almost smallest supertree, based upon iteratively employing tree alignment. This supertree is a global pattern, that can be used both for descriptive and predictive data mining tasks. Experiments performed on two real datasets, show that our approach leads to a drastic compression of the database. Furthermore, when the resulting pattern is used for classification, the results show a considerable improvement over existing algorithms. Moreover, the incremental nature of the algorithm provides a flexible way of dealing with extension or reduction of the original dataset. Finally, the computation of the almost smallest supertree can be easily parallelized.

1 Introduction

Data that can conceptually be viewed as tree structured data abounds in domains such as bio-informatics, web log analysis and XML databases. Important research question in the field of knowledge discovery and data mining involves the construction of descriptive and predictive models for tree structured data. Currently, the dominating approach to mine tree structured data is the class of frequent tree mining algorithms [3, 20]. A major drawback of this approach is that in order to find interesting patterns, the threshold used leads to an explosion in the number of frequent patterns. As a result, the size of the frequent pattern set is often multiple times the size of the original database. Another disadvantage of the frequent pattern mining approach is the assumption that the data resides in a static database, i.e. any addition or deletion of data may result in invalid description and prediction models and hence the mining algorithm must be started again from scratch.

In this work we present a novel method to mine tree structured data, based upon finding an almost smallest supertree. Given a forest, a smallest supertree is a tree of which each tree in the forest is a subtree and there is no tree with less nodes that is also a supertree of every tree in the database. Clearly, given such a smallest supertree, it is the shortest description of the database and, in accordance with the MDL (Minimal Description Length) principle the best model for the dataset [17]. Or in other words, as stated in [12]: several core aspects of datamining are all essentially related to compression. However, even for a set of five sequences with an alphabet size of two, the problem of finding the smallest super sequence is NP-Complete [18]. Since a sequence is a constrained tree, the problem of computing a smallest supertree is also NP-Complete. A smallest supertree of two ordered trees can be derived from the optimal alignment of these trees, with a particular cost function associated to the tree edit operations [16]. As such, deriving the smallest supertree is a special case of tree alignment which in turn is a restricted case of the smallest tree edit distance [22] between two trees. In this work we perform the alignment of multiple trees, which extends the optimal alignment of two trees with heuristics. Although our approach does not lead to the smallest super tree in general, the results show:

- The size of the resulting pattern (in terms of number of nodes) varies (depending on the dataset used) between 2.1% and 22.6% of the original dataset.
- The classifier constructed from the almost minimal supertree shows considerable improvement over existing algorithms to classify tree structured data.
- The incremental nature of the algorithm to construct the almost minimal supertree leads to great flexibility and the possibility to perform parallel computing.

This paper is organized as follows: in the next section, we describe the related work in more detail. In section 3 we introduce the basic concepts of trees and

*Supported by the Netherlands Organisation for Scientific Research (NWO) under grant no. 612.066.304.

introduce the notations used in this paper. Furthermore, we briefly introduce tree alignment and discuss some properties of aligned trees. In the following section we describe our method named MASS, for multiple tree alignment. In section 5 we perform the algorithm on two real datasets and experimentally compare it with frequent tree mining algorithms. Furthermore, we compare the classification performance of the classifier constructed from the smallest supertree and standard classification methods for tree structured data. In the final section we draw conclusions and give directions for further research.

2 Related Work

We already briefly mentioned the class of frequent tree mining algorithms to mine tree structured data. Frequent tree mining can be seen as an extension of the Apriori algorithm [1], to handle tree structured data in the mining process. In short, given a set of tree structured data, the problem is to find all subtrees that satisfy the minimum support constraint, that is, all subtrees that occur in at least $n\%$ of the data records. In order to restrain the solution set several proposals have been conducted. Chi et. al [8] describe an algorithm to mine closed subtrees. Although the algorithm has the advantage that the pruning of non-closed trees is resolved in the mining process—as opposed to post pruning—it has the disadvantage that the resulting pattern set is still enormous. A completely different approach is taken in the work by Siebes et. al [19] for frequent itemsets and the extension by Bathoorn et. al [4] for structured data. They select frequent patterns according to the MDL principle: the smallest set of patterns that best compresses the database. The analogy with our work is that both approaches compress the database, in order to derive meaningful results. However, the difference is that they use local frequent patterns, while our method results in one global pattern that compresses the database directly.

A different class of mining algorithms for tree structured data uses tree edit distance [22] or tree alignment [16] as distance measure; see for example [15, 2]. The general method works as follows: firstly, the distance measure used is computed between each pair of trees in the database; secondly to perform the data mining task, a standard clustering or classification algorithm is used on the computed distances. A major disadvantage of this method is that the distance has to be computed for each pair of trees; hence it is impractical to perform this for medium sized databases and undoable for large realistic databases. Furthermore, the results do not deliver a descriptive model of the data.

Finally, Subdue [9] is a data mining system suited for mining substructures from a single graph. Subdue uses MDL to find substructures that best compress the input graph. This is accomplished by a beam search that iteratively extends substructures—those that give the best compression on the database—with a single node. Besides that our algorithm works on a set of trees and Subdue on a single graph, the main difference is that Subdue uses local optima to achieve a better compression, while our approach uses the global optimum between two trees. So for example, when a substructure compresses the database reasonably well, in Subdue's approach this substructure can only be extended with a node adjacent to the substructure in the original graph. In our approach, when a part of the previously constructed supertree compresses some unseen tree pretty well, other parts of that supertree even when they are not directly connected to the subtree) can also contribute to a yet better compression.

3 Preliminaries

3.1 Tree Basics In this section we provide the basic concepts and notation used in this paper. A labeled rooted ordered tree $T = \{V, E, \leq, L, v_0, \Sigma\}$ is an acyclic directed connected graph which contains a set of nodes V, and an edge set E. The labeling function L is defined as $L : V \to \Sigma$, i.e. L assigns labels from alphabet Σ to nodes in V. The special node v_{root} is called the root of the tree. If $(u, v) \in E$ then u is the parent of v and v is a child of u. For a node v, any node u on the path from the root node to v is called an ancestor of v. If u is an ancestor of v then v is called a descendant of u. Furthermore there is a binary relation '$\leq$' $\subset V^2$ that represents an ordering among siblings.

The size of a tree is defined as the number of nodes it contains and is denoted by $|T|$. Finally, the preorder of a node is the node index according to the preorder traversal.

Definition 1. *Given two labeled rooted trees T_1 and T_2 we call T_2 an* embedded *subtree of T_1 and T_1 an* embedded *supertree of T_2, denoted by $T_2 \preceq_e T_1$, if there exists an injective matching function Φ of V_{T_2} into V_{T_1} satisfying the following conditions for any $v, w \in V_{T_2}$:*

1. *Φ preserves the labels: $L_{T_2}(v) = L_{T_1}(\Phi(v))$.*
2. *Φ preserves the left to right order of the nodes: if* $\text{preorder}(v) < \text{preorder}(w)$ *then* $\text{preorder}(\Phi(v)) < \text{preorder}(\Phi(w))$.
3. *Φ preserves the ancestor-descendant relation: if v is an ancestor of w in T_2 then $\Phi(v)$ is an ancestor of $\Phi(w)$ in T_1.*

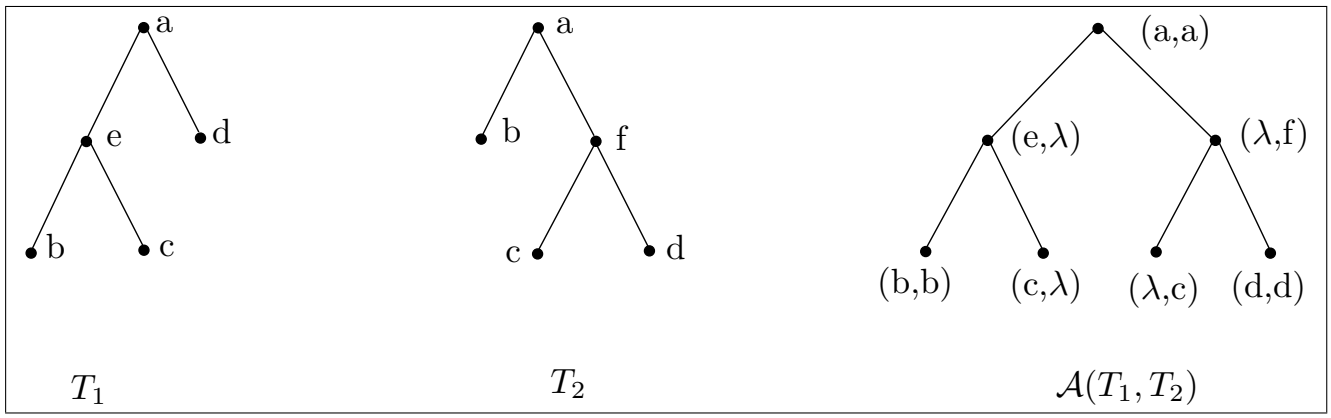

Figure 1: T_1, T_2 and the optimal alignment between T_1 and T_2.

Let $D = \{d_1, \ldots, d_m\}$ denote a database where each $d_i \in D$ is a labeled rooted ordered tree. A tree T is a smallest supertree over the forest D, if $d_i \preceq_e T$, for $1 \leq i \leq m$ and $\forall T', 1 \leq i \leq m : d_i \preceq_e T' \rightarrow |T'| \geq |T|$.

3.2 Alignment Algorithm An alignment of two trees T_1 and T_2 is obtained in the following way: First nodes labeled with spaces (λ with $\lambda \notin \Sigma$) are inserted into both T_1 and T_2, such that the structure of T_1 and T_2 becomes identical. Second, the altered trees are overlaid on each other. Given a score function (μ), that is a function that determines the cost between each pair of nodes, the value of an alignment (or distance) equals the sum of scores. An optimal alignment is an alignment that minimizes the value of an alignment over all possible alignments. An example of an optimal alignment is shown in figure 1, where the alignment distance with score function (3.1) equals 4.

Let v be a node of T_1 and w be a node of T_2, i.e. $v \in V_{T_1}$ and $w \in V_{T_2}$. Denote the subtree of T_1 rooted at v as $T_1[v]$ and similarly the subtree of T_2 rooted at w as $T_2[w]$. Suppose that v has n children ($v_1 \leq v_2 \leq \ldots \leq v_n$) and w has m children ($w_1 \leq w_2 \leq \ldots \leq w_m$). $F_1[v_1, v_n]$ denotes the forest of the subtrees $T_1[v_1], \ldots, T_1[v_n]$ and similarly $F_2[w_1, w_m]$ denotes the forest of the subtrees $T_2[w_1], \ldots, T_1[w_m]$. Furthermore, let Δ denote the alignment distance and $\mathcal{A}$ denote the optimal alignment, both notations are used for trees, nodes and ordered forests.

In order to compute the optimal alignment between T_1 and T_2, the algorithm computes for each pair of nodes v and w: $\Delta(T_1[v], T_2[w_i])$ for all $1 \leq i \leq m$ and $\Delta(T_1[v_i], T_2[w])$ for all $1 \leq i \leq n$. That is, for each pair of nodes v, w the subtree rooted at v should be aligned with each subtree rooted by a child of w and vice versa. Furthermore, also the optimal alignment between the children of two nodes have to be computed, so for each pair of nodes v and w: $\Delta(F_1[v_1, v_n], F_2[w_s, w_t])$ for all $1 \leq s \leq t \leq m$ and $\Delta(F_1[v_s, v_t], F_2[w_1, w_m])$ for all $1 \leq s \leq t \leq n$. That is, for each pair of nodes v, w, the forest that consists of the trees rooted as child of v should be aligned with each consecutive sub forest from the trees rooted as child of w and vice versa. The algorithm, which is a combinatorial optimization and is solved by using a dynamic programming approach, is described in more detail in [16].

As a result, the computation time is $O(|T_1| \times |T_2| \times (\deg(T_1) + \deg(T_2))^2)$. Where $\deg(T_1)$ is the degree of T_1, i.e. the maximal number of children of any node in the tree; correspondingly $\deg(T_2)$ denotes the degree of T_2. In case of unordered trees, the optimal alignment should, for every pair of subtrees $T_1[v]$ and $T_2[w]$, be computed between the children of v and each possible (non-empty) combination between the children of w and vice versa. As a result, alignment of unordered trees is NP-hard, see [16] for more details.

In order to derive a smallest supertree, the following score function is used [16]:

$$\begin{aligned} \mu(v, w) &= 0 \quad |L(v) == L(w) \\ \mu(v, w) &= 1 \quad |L(v) == \lambda \text{ or } L(w) == \lambda \\ \mu(v, w) &= 2 \quad |L(v) \neq L(w) \text{ and } L(v), L(w) \neq \lambda \end{aligned} \tag{3.1}$$

From the minimal alignment distance the optimal alignment can be easily constructed, by tracing back the performed operations. We say there is a match between a node v from T_1 and a node w from T_2 if in the optimal alignment, the nodes are mapped onto each other and the cost of the mapping equals zero, i.e. $\mu(v, w) = 0$. For example in figure 1, the set of matches in $\mathcal{A}(T_1, T_2)$ equals $\{(a, a), (b, b), (d, d)\}$.

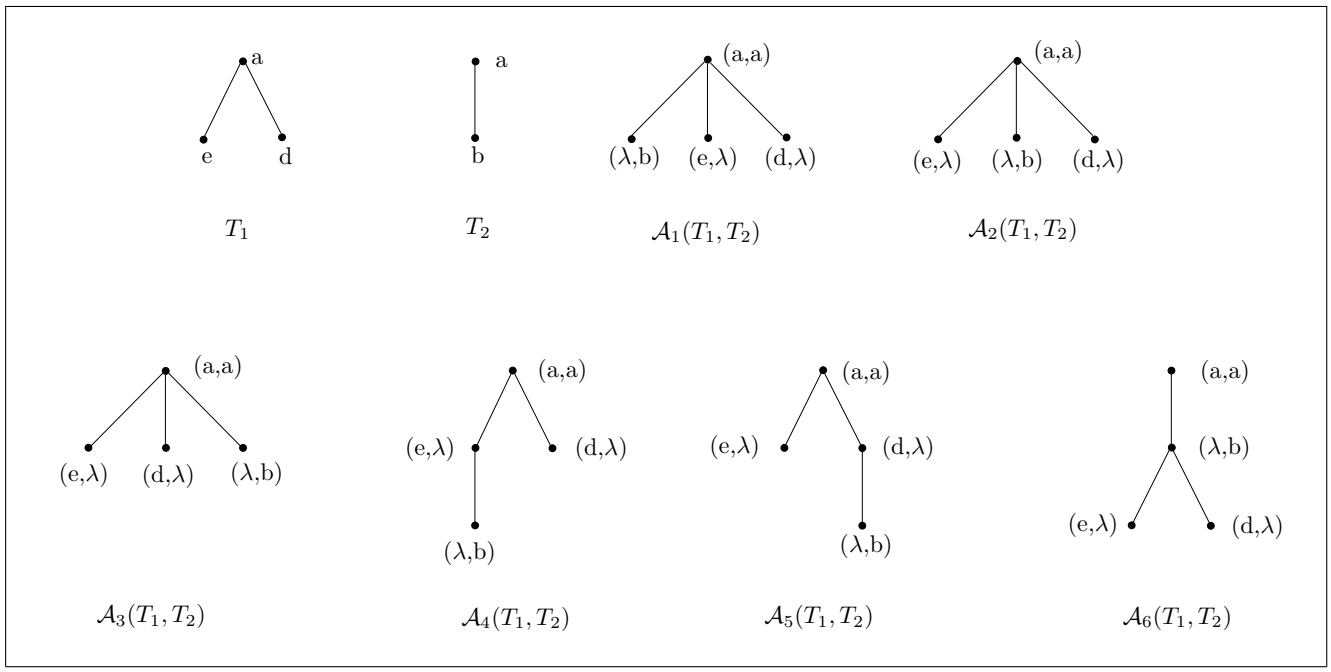

Figure 2: T_1, T_2 and the resulting six optimal alignments between T_1 and T_2.

3.3 Properties of Tree Alignment The alignment of two trees T_1, T_2 results in a tree of which the size is bounded by:

$$\max(|T_1|, |T_2|) \leq |\mathcal{A}(T_1, T_2)| \leq |T_1| + |T_2|.$$

The lower bound occurs when $T_1 \preceq_e T_2$ or $T_2 \preceq_e T_1$, i.e. when all labels of T_1 match with a label in T_2 or vice versa. The worst case occurs when T_1 and T_2 do not have a single node in common.

In case of a unique optimal alignment, the order among the siblings and the ancestor/descendant relation is imposed by the optimal alignment. For example, the optimal alignment of the trees in figure 1, results in a tree for which $e \leq_{\mathcal{A}(T_1,T_2)} f$ but neither $e \leq_{T_1} f$ nor $e \leq_{T_2} f$ holds. Note that the order imposed is only on previously unrelated nodes. Hence the ancestor/descendant relation between nodes in the original trees and the left to right order remain valid, i.e. $T_1 \preceq_e \mathcal{A}(T_1, T_2)$ and $T_2 \preceq_e \mathcal{A}(T_1, T_2)$. Often there is no unique optimal alignment between two trees, but there are multiple optimal alignments instead. We distinguish two causes of non-uniqueness: first when there is no match between a node of the first (sub)tree and a node of the second (sub)tree, and second when there are multiple matches between nodes. For the first case, assume that in order to obtain an optimal alignment $T_1[v]$ and $T_2[w]$ should be aligned. When there is not a single match between any node of $T_1[v]$ and a node of $T_2[w]$ then different alignments result in (expressed in term of $T_2[w]$): v can be added as a child or parent of any node in $T_2[w]$ and the children of v should be aligned with the altered tree rooted at v. For example, in figure 2 for the node labeled b of T_2 ($T_2[b]$) there is no match with a node in $F_1[e, d]$. As a result, all displayed alignments in figure 2 are optimal. The second case arises when a node of $T_1[v]$ matches with multiple nodes in $T_2[w]$ or when different node of $T_1[v]$ match with multiple nodes in $T_2[w]$ and the alignment distance is equal. In this case, the number of optimal alignments is limited to the number of equivalent matches (equivalent in terms of alignment distance). As an example, consider figure 3, the node labeled c of T_2 matches with two nodes in T_1. As a result, the two different optimal alignments are shown.

4 The Mining Algorithm

4.1 Approximate Multiple Tree Alignment The idea underlying the construction of the almost smallest supertree is straightforward: incrementally align all the trees in the database, that is first two trees are aligned, then the resulting optimal alignment is aligned with the next tree, until all trees in the database are processed. However, as noticed earlier, the optimal alignment between two trees does not have to be uniquely defined. So, given that the alignment of the first two trees results in multiple optimal alignments, the questions

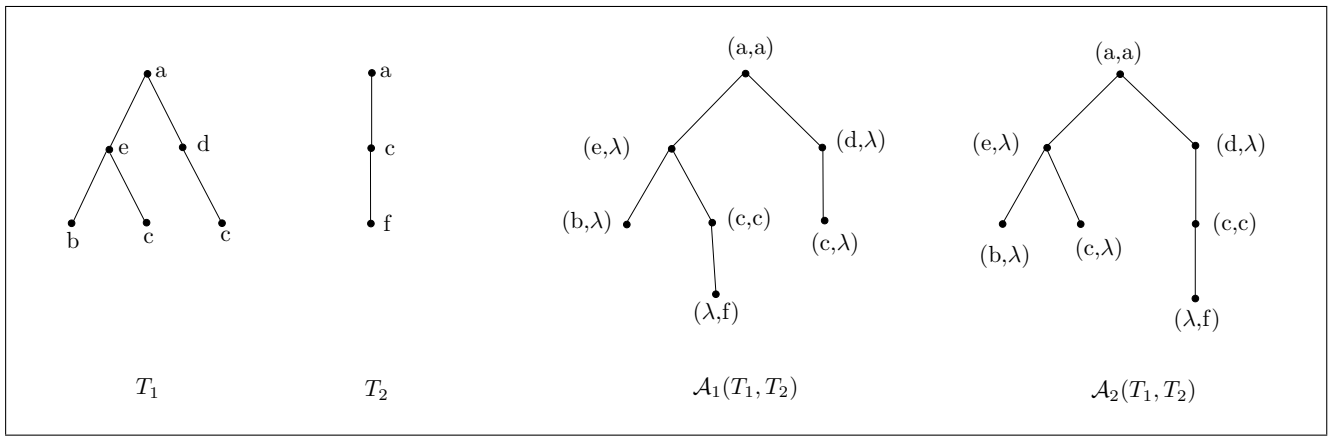

Figure 3: T_1, T_2 and the resulting two optimal alignments between T_1 and T_2.

arises which of these alignments should be aligned with the next tree in the database. Clearly, from a computational point of view it is undoable to align all optimal alignments with the rest of the database. To tackle this problem, we use a heuristic that chooses the most likely alignment. To do this, we add a support value to nodes in the aligned tree, that keeps track of the number of matches in all preceding alignments. So, initially the support value of the nodes equals one. After the first optimal alignment the support value for every node in the set of matches is increased by one, and so on. In case there are multiple equivalent optimal alignments, we choose the optimal alignment that maximizes the sum of the support values for the set of matches. Obviously, this heuristic only makes sense when the presence of multiple optimal alignments is caused by multiple match labels. Clearly, in case the multiple optimal alignments are caused by the lack of matches between the two (sub) trees, there is no prior knowledge of more or less likely optimal alignment. This case is handled by the algorithm, by choosing one of the optimal alignments in a uniform manner. Notice, that MASS is a non-deterministic algorithm: the supertree obtained is depended on the order in which the trees from the forest are processed. In order to obtain a lossless compression, the algorithm have to keep track, to which trees in the forest the nodes in the supertree belong.

From a theoretical point of view, we may assume that the trees from the forest are embedded subtrees sampled from one mastertree, which is for example the case in the synthetic tree structured database generators used in [8, 7, 20]. Given that we have a sufficient amount of data, we expect the smallest supertree to be a good estimator of the unknown mastertree. However, because our method is an approximation, some errors are likely to be introduced. To overcome this, when all data is processed, we prune the nodes that have a low support value. This threshold is given as input parameter; typical values we used in the experiments are within range of 0.2% till 0.5%. The pruning is done as follows: when a node has a support under the predefined treshold then, the node is deleted and all its childeren and their decendants are added as childeren of the corresponding parent from the deleted node. Evidently, the pruned supertree is no longer a supertree of all trees in the forest.

From the algorithmic description given earlier, it is immediately clear how to achieve speedup by means of parallelization: split the dataset in n equal parts and compute for each part the almost smallest supertree, finally align these n supertrees to each other. The incremental approach further has the advantage that data can be added to or removed from the almost smallest supertree, i.e. when the underlying data distribution is changing over time (concept drift) new data can be added and data not longer representative for the underlying data distribution can be removed. The deletion of data from the supertree can be performed as follows: first the optimal alignment between the model and data to be deleted is computed, secondly for each node in the match set of the optimal alignment the support value is decreased by one. Optionally the resulting tree can be pruned. When a smallest supertree is aligned with another smallest supertree, the support value of the nodes in the match sets are added or subtracted (depending on the goal of adding or removing data) from the corresponding nodes that match.

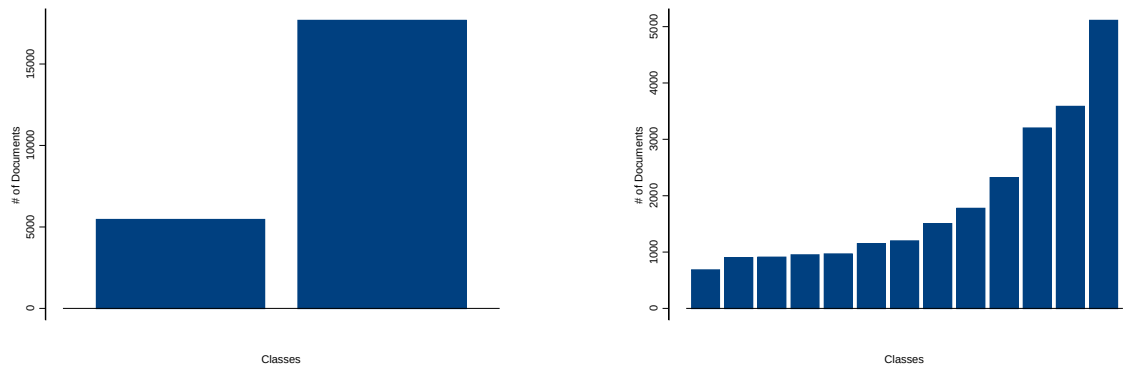

Figure 4: Distribution of the data over the different classes, the classes are sorted according to the number of trees they contain. Left the distribution for the CSLOG dataset, right the distribution for the Wikipedia dataset.

5 Experiments

The goal of the experiments is three-fold:

- The first goal is to analyze the mining algorithm, in terms of computation time and compression of the original dataset. We compare these with the run time and size of the patterns of treeminer [20], a frequent tree mining algorithm.
- The second goal is to analyze the convergence of the mining algorithm, i.e. since we assume that when given enough samples of the data, the underlying mastertree might be captured. The question arises: does this occur in the used dataset and if so how many samples are needed to accomplish this ?
- The third goal is to compare the predictive performance of MASS with a state of the art tree structured classification algorithm Xrules [21] and a more straightforward approach that uses the frequent patterns as binary features in decision trees. The performance measures used are accuracy and area under the ROC curve.

5.1 Data Sets We performed the experiments on two real datasets namely the CSLOG dataset created and used by Zaki and Aggarwal [21] and a subset of the Wikipedia XML dataset [10].

The CSLOG dataset consists of user sessions of the RPI CS website, collected over a period of three weeks. Each user session consists of a graph and contains the websites a user visited on the RPI CS domain. These graphs were transformed to trees by only enabling forward edges starting from the root node. The goal of the classification task is to discriminate between users who come from the edu domain and users from another domain, based upon the user's browsing behavior. In total there are $23,011$ trees in the CSLOG dataset, where the average tree size is 8.02. The relative frequency of the classes is 23.57% for the class that represents the edu domain and 76.43% for the other class, graphically shown in figure 4.

The Wikipedia XML dataset was provided for the document mining track at INEX 2006. The collection consists of $150,094$ XML documents with 60 different class labels. The collection was constructed by converting web pages from the Wikipedia project to XML documents. The class labels correspond to the Wikipedia portal categories of the web pages. In order to compare the results with frequent tree mining algorithms, we selected only those documents where the number of nodes in the tree was less than 20. The reason for selecting only the smaller trees, is that treeminer [20] is not able to run the entire dataset with the support threshold set to 0.6% on a server with 8GB of main memory available. The reason for this is described in [7]: in the worst case, for the embedded subtree mining algorithm the scopelist size is exponential in the size of the data tree. Additionally, we solely used documents that belong to classes that contained more than 400 documents. The resulting reduced dataset consists of $24,222$ documents that are distributed over 13 classes, the distribution is shown in figure 4; the average size of the trees in the database equals 13.95.

All the reported measures where obtained on the dataset using ten-fold cross validation.

5.2 Compression and Runtime The first experiment is designed to evaluate and compare MASS with regard to both run time and compression of the database. These results are compared with the run

		Database	MASS	$\mathbf{MASS_p}$	$\mathbf{treeminer}_{p1}$	$\mathbf{treeminer}_{p2}$
Wikipedia Dataset						
	Data size	$307,125$	$6,452$	$2,815$	$4.15232 * 10^7$	$1,080,194$
	Runtime		$1,159$		$1,201$	968
CSLOG Dataset						
	Data size	$168,531$	$38,137$	274	$1,129,551$	$10,116$
	Runtime		$3,862$		33	2

Table 1: Compression and run time results for the two datasets used.

time and size of the resulting pattern set, as obtained from the frequent tree mining algorithm treeminer [20]. We have chosen to compare with this particular tree mining algorithm because MASS and treeminer use the same tree inclusion relation, as opposed to for example FREQT [3]. Contrary to the more common notion to define the size of the forest as the number of trees it contains, we defined it as the sum of the sizes of the trees in the forest, where the size of a tree equals the number of nodes it contains. For the frequent tree mining algorithm, we used the minimum support values that were optimal in the experiments to construct a classifier from the frequent pattern set, as conducted in the subsection 5.4. For the MASS algorithm we present the results both for the pruned and the unpruned version, also in this case the optimal minimal support value is determined according to the best obtained results in subsection 5.4. The threshold used for the pruned version of MASS ($\mathbf{MASS_p}$) was set to 0.2% for the two datasets. The frequent tree mining algorithm was performed on both datasets with the minimum support treshold set to both 0.2% ($\mathbf{treeminer}_{p1}$) and 0.5% for the CSLOG dataset and 0.6% for the Wikipedia dataset ($\mathbf{treeminer}_{p2}$).

Because the datasets consist of multiple classes, the datasets were first split according to their class. Secondly, MASS was applied to the different parts of the dataset. The results reported are aggregated over the different classes. For the frequent tree mining algorithm, the minimum support constraint was applied per class, i.e. a pattern was frequent if it occurred in at least one class in n% of the records, with n being the minimum support threshold. Furthermore, patterns that were frequent in multiple classes, contributed only once to the size of the frequent pattern set.

As the results in table 1 show, the size of the supertree for the Wikipedia dataset is about 2.1% of the original data set size. Hence, the database and the frequent pattern set can be described by one pattern that has a substantially smaller size. Comparing the models that give optimal predicitive performance, the pruned version of the supertree is a factor between 14750, and 380 times smaller than the size of the frequent pattern set. The run time performance of MASS on the Wikipedia dataset is slightly better than the run time for the frequent tree mining algorithm at minimum support threshold 0.2%, while the run time of the later algorithm for which the minimum support treshold set at 0.6% outperforms MASS. For the CSLOG dataset the size of the supertree is about 22.6% of the size of the dataset, which is quite modest compared to the reduction obtained on the Wikipedia dataset. The reason for this difference in reduction is that the CSLOG dataset is very sparse, which also resulted in a relatively small size of the frequent pattern set. Hence, the number of matches between two trees is relatively small, which results in an increasing size of the smallest supertree. Another indicator for the sparseness of the dataset is that the size of the pruned version is about 139 times smaller than the unpruned supertree, i.e. there are quite a few nodes that occur with a support value less than 0.2% in the smallest supertree. Because the number of matches between two trees is relatively small the run time for MASS is high. Recall, from the computional complexity as stated in subsection 3.2, that the complexcity depends on the size of the trees. Since in the CSLOG dataset the supertree grows larger at each iteration, the run time also increases with each iteration. Finally, the size of the pruned supertree is between 4122 and 37 times smaller than the size of the frequent pattern sets.

Summarizing, for both datasets the almost smallest supertree reduces the datasets drastically, but the reduction obtained depends on the density of the dataset. Also the pruned supertrees are far smaller in size and, hence, easier to comprehend for a human than the frequent pattern sets. Finally, the time needed to compute the almost smallest supertree depends on the dataset but, in comparison with the computation time to deliver the frequent patterns, is relatively high.

5.3 Convergence Analysis The next question to answer is: does the smallest supertree converge to a model from which the data is sampled. Secondly, if so,

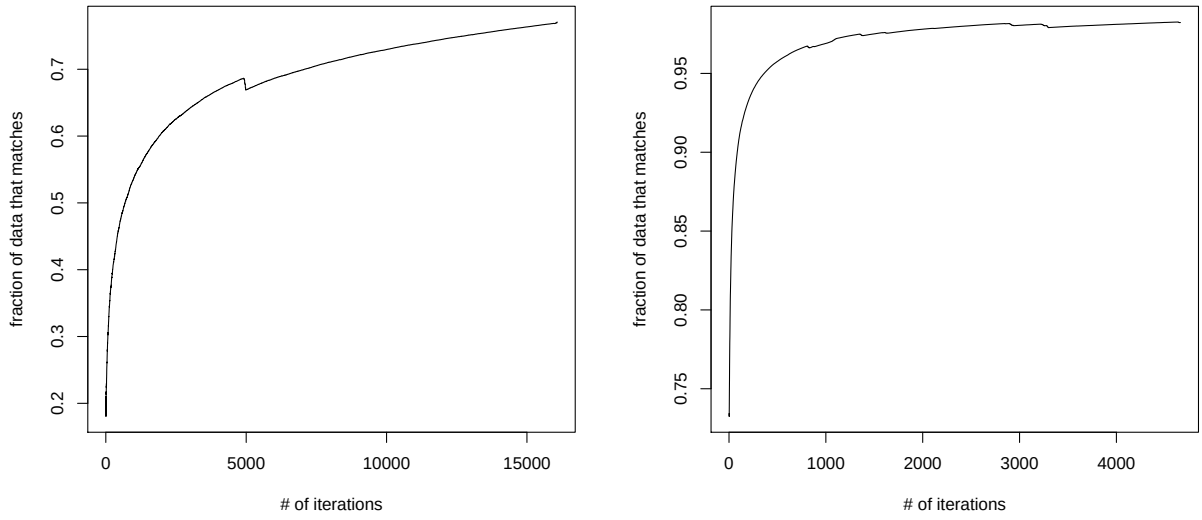

Figure 5: Fraction of the number of nodes in the match set and the tree size, for a supertree incrementally constructed. Left shows the plot for CSLOG dataset, right for the Wikipedia dataset.

how fast does it converge ? Given a sufficient amount of data, in the ideal situation the MASS algorithm should produce a supertree that has the property that the alignment of unseen data with the supertree results in a near perfect match, i.e. almost all nodes of the unseen data match to nodes of the supertree. To analyze this, we randomized the order of the trees and consecutively align the remaining trees of the database, where we keep track of the number of nodes in the match set and the number of nodes in the tree that is aligned to the supertree. This fraction, along with the number of trees already processed, is shown in figure 5. The results are averaged over all classes and over all ten-folds, i.e. the fraction of matches of the nth tree of class 1, fold 1 is averaged with the fraction of matches of the nth tree of all classes and all folds, etc. However, not every class and not every fold contains an equal number of trees and hence the results obtained in later iterations are averaged over less examples then the results obtained in the beginning.

The results for the CSLOG dataset show that initially the overlap is quite low. However, when the number of trees processed increases, the overlap between the supertree and the unprocessed data increases drastically too. But still, the ideal situation does not occur for the CSLOG dataset. Since the fraction is still increasing when data are being added, it seems likely that the addition of more data would result in a desired ratio that lies above 0.95. Remarkable for the CSLOG dataset is the kink in the plot that occurs slightly after 5000 iteration. This is most likely because there are slightly more trees then 5000 in the "edu" class, and apparently the trees from this class have more overlap then the trees from the other class. This makes sense, since "users from the edu domain" is a much more specific characterization then "users from an other domain" (the other class). For the Wikipedia dataset, the overlap initially starts quite high. As the number of trees processed increases so does the overlap, until it reaches a score of 0.95. After this point the growth flattens and finally reaches a value slightly above 0.98. Hence, the supertrees constructed for the Wikipedia dataset capture the underlying data model quite well and the convergence goes fast.

5.4 Classification So far, the major justification to construct an almost smallest supertree is that the database is described by a pattern that has a far smaller size than the original database. To further evaluate and compare MASS with existing mining algorithms, we constructed a classifier from the derived patterns. The performance of this classifier is compared with Xrules [21] a rule based classifier for tree structured data and decision trees, that uses frequent patterns as binary features. We used both accuracy and area under the ROC curve as performance measure. Accuracy is the the ratio of the number of correctly classified

Database	Measure	MASS	Xrules	Decision Trees
Wikipedia Dataset				
	Accuracy	60.49 ± 0.97	64.12 ± 0.75	64.21 ± 0.96
	Area under the ROC curve	89.77 ± 0.44	85.48 ± 0.48	87.03 ± 0.55
CSLOG Dataset				
	Accuracy	82.18 ± 0.53	81.52 ± 0.57	83.48 ± 0.62
	Area under the ROC curve	79.02 ± 1.27	72.34 ± 1.34	72.63 ± 1.53

Table 2: Classification results for the Wikipedia and CSLOG datasets.

documents and the total number of documents in the test set. Area under the ROC curve is a measure that compares the classification model with a classifier that has random performance [14]. This measure was also computed on the test set. The reason we compare our results with these two classifiers is because we want to compare the quality of the descriptive patterns from the different mining algorithms. And Xrules is, as far as we know, the only method that directly uses the frequent tree structured patterns in the construction of the classifier. However, there is still a wide choice of classification algorithms that could use the frequent patterns as features, for example in [11] support vector machines are used and in [6] decision trees.

Given a dataset $D = \{d_1, \ldots, d_m\}$ consisting of m trees, each tree belongs to exactly one class, where the class label is assigned from the set of class labels $C = \{c_1, \ldots, c_k\}$. With D_{c_i} we denote the set of trees in the database that has class label c_i, likewise with $D_{\overline{c_i}}$ the set of trees in the database is denoted that has a class label different from c_i.

The general idea behind the classifier constructed from the supertree, is that in each supertree a value for the nodes is derived, that corresponds to their discriminating strength. The similarity between a tree T and a class, equals the sum of the discriminating values for nodes in the supertree that are in the match set of the optimal alignment between T and the supertree. In order to construct a predictive model that uses the almost smallest supertree, we performed the following: First, for every class in the dataset, MASS was applied to all data per class, i.e. $\forall D_{c_i}$ MASS($\mathrm{D_{c_i}}$), for $1 \leq i \leq k$. For each of the k supertrees derived (say $S_1, \ldots S_k$), we pruned the supertree with an user supplied minimum support value. Also to each node in the supertree a negative weight variable was assigned, initialized at 0. Subsequently, we randomly selected trees from the database that do not belong to the class of the current supertree and subtracted these from the supertree. So, for example for supertree S_i that was constructed by aligning the data from D_{c_i}, we selected about $1/4 \times |D_{c_i}|$ trees from $D_{\overline{c_i}}$. These selected trees are aligned with the supertree S_i, where for each node from S_i that is in the match set of the optimal alignment, the negative weight counter is increased by one. As a result, we now have two counts associated to each node in the supertree. The idea is to use these counts, to determine which parts are discriminating in the supertree, such that a match with the discriminating parts results in a high score. To achieve this, we first re-weight the support and the negative weight such that both values are in the interval $[0 \ldots 1]$, i.e. support = support$/|D_{ci}|$ and negative weight = $4 *$ negative weight$/|D_{ci}|$. Next, we determined "the discriminating strength" for each node in the supertree. This was estimated as the fraction of the scaled support and the scaled negative weight; if the scaled negative weight equals zero then it is replaced with 0.01. To the nodes in the supertree that have a discriminating strength strictly above 1, we will further refer as the discriminating set of nodes. Finally, the prediction of a class for a tree T works as follows: T is aligned with each supertree derived. From these optimal alignments we compute the sum of the discriminating strength from the nodes that were both in the match set and in the discriminating set of nodes. This score is further adjusted to compensate for the size of the supertree, i.e. larger trees and in particular trees that contain a larger discriminating set have a higher probability to achieve a higher score. To solve this, the computed score is divided by the logarithm of the size of the discriminating set in the supertree. As a result, we obtain for each class in the dataset a score that measures the similarity between T and the supertree derived from the class. T is predicted to belong to the class for which the similarity score with the corresponding supertree is maximal.

Both Xrules and the decision tree constructed by using the frequent patterns as binary features use discriminative patterns, i.e. patterns that discriminate between the different classes. To derive these discriminating patterns, first treeminer [20] derives all frequent patterns within a class. Secondly, from all frequent patterns within a class, those patterns that are good descriptors of the class are selected. This is done by means of de-

termining the confidence $P(c_i|T)$ of a pattern T, that is the probability of a class given the pattern. If this confidence is greater than 0.5, then the pattern is regarded as a good discriminator for its class.

Xrules now proceeds by ordering the patterns according to the highest confidence score. If the confidence score between two trees is equal, then the patterns are further sorted according to the support value in the database and the size of the tree, respectively. The classification of a tree T works as follows: T is assigned to the class of the first pattern in the ordered pattern set that is a subtree of T. If none of the patterns is a subtree of T, the class predicted for T is the default class; that is, the class that contained the most elements in the training set.

The decision tree uses the set of discriminating patterns as binary features, i.e. each feature indicates the presence or absence of a discriminating pattern in a record. We used the implementation of decision trees provided by Borgelt [5]. This implementation uses a top down divide and conquer technique to build the decision tree. At each node in the decision tree, an attribute is selected—in a greedy manner—that best discriminates between the classes. As selection criterion, information gain was used. Additionally, after construction of the decision tree we used confidence level pruning to avoid overfitting. The parameter used with confidence level pruning was set to 50%.

Note that, besides the minimum support constraint, we did not optimize for other parameters. For example, for the confidence of a pattern we used 0.5 as a threshold in the Xrules and decision tree classifier, which is in accordance with the best achieved results for Xrules over the CSLOG dataset [21]. Also, the classification algorithm constructed from the almost smallest supertree is presumably suboptimal; we tried some parameter settings and picked the ones that achieved the best results for the training set. It should be noted that our goal is not to build the "best" classification method for tree structured data, but to compare the quality of the derived patterns.

The classification results are shown in table 2. The minimum support threshold for MASS and Xrules for both datasets, was set to 0.2%, while the minimum support value used for the frequent patterns that were used by decision trees was set to 0.5% for the CSLOG dataset and 0.6% for the Wikipedia dataset. All scores obtained are averaged over the ten folds. For the Wikipedia dataset, the accuracy estimate for the classifier constructed from the supertrees is considerably below the accuracy estimates for the other two classification methods. Compared with Xrules, the difference is significant. However, concerning the area under the ROC curve estimates, the score obtained by our method considerably outperforms the other two classification methods. Notice, the remarkable discrepancy between the accuracy and the AUC estimates on the Wikipedia dataset. A closer investigation, revealed that for some trees in the dataset, the classifier constructed from the supertrees, assigned high similarity scores for the tree to which the class belong and a slightly higher score for another class. This likely caused the lower accuracy value, while remaining a fairly high AUC value on the Wikipedia dataset.

Also for the CSLOG dataset, our method achieves considerably better results in terms of area under the curve than both Xrules and decision trees. The obtained accuracy estimates of our method is in between the estimate obtained by Xrules and the estimate obtained by the decision trees, with no significant difference between any of the classification methods used.

Concluding, classification results obtained by a classifier constructed from the almost smallest supertrees are, in terms of area under the ROC curve, considerably better then the results obtained by state of the art classification methods, based upon frequent patterns. This also shows that essential information from the dataset is captured in the pruned, almost smallest supertrees. Hence, in terms of predictive performance, the almost smallest supertree is a competing alternative for frequent patterns. However, when the classification methods are compared in terms of accuracy, the classifier constructed from the almost smallest supertrees does not achieve the best results in all cases. The question remains if the performance of the classifier can be increased in terms of accuracy while the performance in terms of area under the curve remains at least stable.

6 Conclusion

In this paper we introduced MASS: a new mining algorithm for tree structured data, based upon incrementally aligning trees to each other. The almost smallest supertree constructed by MASS results in a drastic compression of the original dataset. Moreover, the classification results strongly suggest that essential information of the dataset is captured by the pruned version of the almost smallest supertree. The greatest disadvantage of MASS is its required computation time, although this can be reduced using multiple processors. Moreover, MASS has the desirable property that once the almost smallest supertree is constructed data can be added or removed. Further research includes the development of more advanced classifiers from the almost smallest supertrees and heuristics to speed op the mining process. Furthermore, exhaustive investigation of the convergence of the MASS algorithm, with data

sampled from a known data distribution, is a topic we plan to investigate in the near future.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499, 1994.

[2] K. Aoki, A. Yamaguchi, Y. Okuno, T. Akutsu, N. Ueda, M. Kanehisa, and H. Mamitsuka. Efficient tree-matching methods for accurate carbohydrate database queries. *Genome Informatics*, 14:134–143, 2003.

[3] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *SIAM International Conference on Data Mining*, 2002.

[4] R. Bathoorn, A.Koopman, and A. Siebes. Reducing the frequent pattern set. In *Workshops Proceedings of (ICDM 2006)*, pages 55–59, 2006.

[5] C. Borgelt. A decision tree plug-in for dataengine. In *Proc. 6th European Congress on Intelligent Techniques and Soft Computing*, 1998.

[6] B. Bringmann and A. Zimmermann. Tree^2 - decision trees for tree structured data. In *European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 46–58, 2005.

[7] Y. Chi, R. Muntz, S. Nijssen, and J. Kok. Frequent subtree mining - an overview. *Fundamenta Informaticae.*, 66(1-2):161–198, 2005.

[8] Y. Chi, Y. Yang, Y. Xia, and R. R. Muntz. Cmtreeminer: Mining both closed and maximal frequent subtrees. In *PAKDD'04*, May 2004.

[9] D. Cook and L. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[10] L. Denoyer and P. Gallinari. The Wikipedia XML Corpus. *SIGIR Forum*, 2006.

[11] M. Deshpande, M. Kuramochi, and G. Karypis. Frequent sub-structure-based approaches for classifying chemical compounds. In *IEE International Conference on Data Mining*, 2003.

[12] C. Faloutsos and V. Megalooikonomou. On data mining, compression, and kolmogorov complexity. *Data Min. Knowl. Discov.*, 15(1):3–20, 2007.

[13] M. Garofalakis and A. Kumar. Correlating xml data streams using tree-edit distance embeddings. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 143–154, 2003.

[14] D. J. Hand and R. J. Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning*, 45(2):171–186, 2001.

[15] A. Hemnani and S. Bressan. Information extraction - tree alignment approach to pattern discovery in web documents. In *DEXA '02: Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 789–798, 2002.

[16] T. Jiang, L. Wang, and K. Zhang. Alignment of trees: an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148, 1995.

[17] M. Li and P. M. B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1993.

[18] D. Maier. The complexity of some problems on subsequences and supersequences. *J. of the ACM*, 25(2):322–336, 1978.

[19] A. Siebes, J. Vreeken, and M van Leeuwen. Item sets that compress. In J. Ghosh, D. Lambert, D. Skillicorn, and J. Srivastava, editors, *SIAM International Conference on Data Mining*, 2006.

[20] M. J. Zaki. Efficiently mining frequent trees in a forest. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2002.

[21] M. J. Zaki and C. C. Aggarwal. Xrules: an effective structural classifier for XML data. In L. Getoor, T. E. Senator, P. Domingos, and C. Faloutsos, editors, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 316–325, 2003.

[22] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.