Efficient Extraction of Maximally Common Subtrees from XML Documents for Web Services

Juryon Paik*, Young J. Son†, Farshad Fouthoi† and Ungmo Kim*
*Department of Computer Engineering, Sungkyunkwan University
300 Chunchun-dong, Jangan-gu, Suwon,
Gyeonggi-do 440-746, Republic of Korea
Email: {wise96, umkim}@ece.skku.ac.kr
†Dept. of Computer Science
Wayne State University, Detroit, MI, USA

Abstract— **Web services need to integrate and classify XML documents received from multiple and heterogeneous sources. To this end, it requires a mechanism for extracting common structures from a large XML dataset, called frequent subtrees. In this paper we propose an efficient and scalable algorithm, EMaxS, for mining frequent subtrees of Web XML documents stored in web servers. Compared with previous works, the proposed algorithm uses only simple bitwise operations and dose not require any join steps which are typically expensive.**

Index Terms— **XML document, XML router, Frequent subtree, XML mining, Frequent pattern discovery**

I. Introduction

Over the past several years there has been a tremendous surge of interest in XML along with Web services. It is strongly believed that XML will be the next '*lingua franca*' for the Web as a major standard for storing and exchanging information. Web services use XML-based open standards, such as the WSDL for service definition and the SOAP for service invocation.

In the standard Web service model, as described in [3], a web server receives XML data from heterogeneous sources, and one or more XML routers classify the received data. A web server plays a role as a data warehouse for Web XML documents. The routers classify the data from the web server according to its type and content, and dispatch the classified data to an appropriate back-end server. After web services are clustered to the specialized back-end servers, end-users or applications can extract a symbolic characterization of each back-end servers. Even more, the extracted characteristics can be used by XML routers to cluster those bulky web services received in a web server. Since XML-based documents are typically represented by labeled tree structures, the most common approach to extract desirable characteristics, mainly called 'information/knowlwdge' in the data mining community, is to find *frequent subtrees* in a large-scaled tree-structured dataset. Most traditional approaches are based on typical iterative processes [5], [11], in which not only the raw data can be mined several times, but also the mined patterns might constitute the starting point for further mining on them. It is required to find more scalable and less burdensome methods for extraction of frequent subtrees.

This research was supported by UTRI funded by the Korean Ministry of Science and Technology.

In this paper, we present a novel method, called *EMaxS (Extract Maximal Subtrees from tree structures)*, that automatically extracts a set of frequent trees occurring as maximal common subtrees embedded in a user-specified sufficient number of trees of a collection. By construction, this method contributes (i) a novel idea of binary coding representation (ii)a simplified process during the phase of generating any frequent subtrees (iii) a characterization of each specialized back-end cluster by a set of frequent subtrees, and (iv) a clustering of the received XML-based services.

The rest of this paper is organized as follows. We begin by reviewing some related works in Section II. We continue in Section III with a description of some definitions used throughout the paper. Then, we present our new approach EMaxS in Section IV, and report experimental results in Section V. Finally, we conclude in Section VI with discussion of future work and goals.

II. Related Work

Web services use open standards based on the Extensible Markup Language [14], such as Web Services Description Language [13] for service definition and the Simple Object Access Protocol [12] for service invocation. WSDL is becoming increasingly popular in advertising service information sources on the Web. Users can reach the services using SOAP.

We believe that, from a research viewpoint, it is useful to consider Web Services as a paradigm for aggregation. Using that analogy, we investigate the challenges researchers have uncovered related to knowledge mining from aggregated information, especially semi-structured XML data.

Only a small number of researches have been done on data mining from semi-structured data. Most related works would be that study of the frequent tree discovery. The original idea of mining from semi-structured data dates back to the classical paper of Wang and Liu [9]. They considered mining of collections of paths in ordered trees with Apriori-style technique. They proposed the mining of wider class of substructures which are subtrees called schemas. Asai et al. [2] presented an efficient algorithm for discovering frequent substructures from

a large collection of semi-structured data. As knowledge representations for tree structured data, a tree-expression pattern [10] and a regular path expression [4] were proposed. In [10], Wang and Liu presented the algorithm for finding maximally frequent tree-expression patterns from semi-structured data. Fernandez and Suciu [4] suggested the algorithm for finding optimal regular path expressions from semi-structured data. Very recently, two methods are introduced, which are string representations and tag tree patterns. For effective subtree generation, Zaki [11] independently proposed efficient algorithms for the frequent pattern discovery problem for ordered trees. He adopted an efficient enumeration technique using a special string representation of ordered trees, while Miyahara et al. [7] proposed the concept of tag tree patterns with unordered children from the view point of the semantics of OEM data. Termier et al. developed a mining algorithm called TreeFinder in [8], which presents a combination of relational descriptions of labeled trees and θ-subsumption, to extract frequent subtrees from a set of trees. Other recent works related to frequent subtree mining include mining frequent graph patterns [5], [6]. Such graph mining algorithms are likely to be too general for tree mining as pointed out in [11].

The problem of previous approaches is that they represent each node label of XML trees as a character string, generating numerous number of strings from which frequent subtrees are identified through a process of repeated pruning, known as the *generate-test* strategy. The round complexity of generate-test algorithm becomes prohibitive on a large number of strings, even when each round requires many expensive join operations. Therefore, as the number of XML documents increases, the efficiency of previously developed algorithms deteriorates rapidly since both the cost of join operations and the number of pruning rounds add up.

III. Background Concepts

In this section, we briefly introduce some notions of tree model for mining XML data.

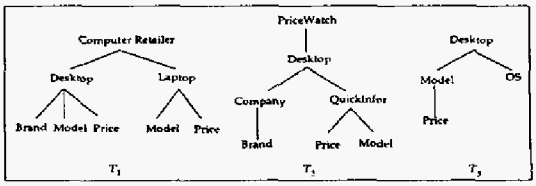
A. XML for Web Services

Web services provide an extensible and interoperable framework for application–to–application communication, thanks to the use of universal data formats, XML. We model a XML document as a labeled tree with a single root.

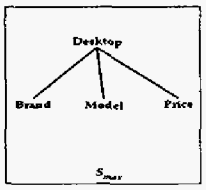
Definition 1 (Labeled Tree): Let $T = (N, E)$ be a tree, where N is a set of nodes, and E is a set of edges. We say that tree $T = (N, E)$ is a *labeled tree*, if there exists a labelling function that assigns a label to each node in N.

Definition 2 (Subtree): We say that a tree $S = (N_S, E_S)$ is a *subtree* of a tree $T = (N, E)$, denoted by $S \preceq T$, if and only if $N_S \subseteq N$ and for all edges $(u, v) \in E_S$, u is an ancestor of v in T.

Note that in the above subtree definition, we require that for any edge (u, v) occurring in a subtree S, there be a path from node u to node v in tree T. This definition extends the conventional definition of a subtree, where for all edges $(u, v) \in E_S$, u must be a parent of v in T. Hence, Definition 2 preserves the ancestor relation but not necessarily the parent relation.



(a) A set of input trees $\mathcal{D}$



(b) The maximal $\frac{2}{3}$-frequent subtree

Fig. 1. Extracted Maximal Frequent Subtree from a database $\mathcal{D}$

B. Problem Statement

Let $\mathcal{D} = \{T_1, T_2, \ldots, T_i\}$ be a database of input trees and let $|\mathcal{D}|$ be the number of trees in $\mathcal{D}$.

Definition 3 (Minimum support): Given a database of trees $\mathcal{D}$, and a subtree S, the frequency of S in $\mathcal{D}$, $freq_{\mathcal{D}}(S) = \Sigma_{T \in \mathcal{D}} freq_T(S)$ where $freq_T(S)$ is 1 if S occurs in T and 0 otherwise. The *support* of S in $\mathcal{D}$, $sup_{\mathcal{D}}(S)$, is the percentage of the trees in $\mathcal{D}$ that contain S, i.e., $sup_{\mathcal{D}}(S) = \frac{freq_{\mathcal{D}}(S)}{|\mathcal{D}|}$. The *minimum support* σ is the percentage which satisfies $sup_{\mathcal{D}}(S) \geq \sigma$.

The above definition excludes multiple occurrences of the subtree in a tree since we consider only whether a tree includes the subtree.

Definition 4 (Maximal Frequent Subtree): Given some minimum support σ, a subtree S is said to be *maximal frequent* iff:

1) the support for S is not less than the required support σ.
2) there is no any other σ-frequent subtree S' in $\mathcal{D}$ such that S is included in S'.

The subtree S is just called a σ-frequent subtree if it only satisfies the first property. Once all maximal σ-frequent subtrees are found, it is straightforward to discover all σ-frequent subtrees. Note that every frequent subtree can be derived from the maximal frequent subtrees by the fact that all subtrees of maximal frequent subtrees are also frequent [1], [7].

Example 1: An example of a set of input trees $\mathcal{D}$ with various nesting of labels is shown in Fig. 1(a). The nesting of `Desktop`, `Model`, and `Price` differs in trees T_1, T_2 and T_3, i.e., the node labeled with *Model* is the direct child of the node labeled with *Desktop* in T_1 and T_3, while there is a node in between the corresponding nodes in T_2. Despite the variations in the structure, we want to find frequently occurring maximal tree pattern shown in Fig. 1(b). The given minimum support $\sigma = \frac{2}{3}$. The example shown in Fig. 1 is used throughout the paper to illustrate how to extract maximal frequent subtrees.

IV. The Proposed Algorithm EMaxS

The problem of extracting maximal frequent subtrees is significantly reduced in our proposed algorithm. This is mainly due to binary code representation of XML documents. It is enough to represent, for instance, 16 node labels only with a 4-bit length code.

In the previous approaches, the inefficiency of extracting frequent subtrees stems from the fact that they represent an element of XML document as a character string. This way of encoding necessarily implies the generation of thousands of string node objects. This causes increase of search space and time exponentially in the number of labels. The problem in this case is that candidate subtrees cannot or not easily be generated without iterative join operations. In our proposed algorithm, the binary coding representation comes to play a key role in solving these potential problems with a new data structure and thus in simplifying the frequent subtree extraction.

A. Key Features

The fundamental idea of algorithm EMaxS is as follows: first every node label is mapped to an n-bit binary code. All trees in a database are, then, represented as sets of bit sequences by concatenations of each n-bit code. Afterwards the frequent sets are mined by a special operation. After all frequent sets are found, maximal frequent subtrees are constructed by incrementally extending those frequent subtrees.

1) Bit Sequence: Typical methods of representing a tree are an adjacency list [5], adjacency matrix [6], or character string [2], [7], [11]. Extracting frequent subtrees by using those methods requires expensive join operations. Thus, to avoid the unnecessary join expense, we adopt binary coding method for representing the tree.

Let $\mathcal{L}$ be a set of labeled nodes in a set of trees $\mathcal{D}$. To represent each tree as a set of bit sequences; firstly, an unique n-bit binary code is randomly assigned to every labeled node. Note that the same n-bit code must be assigned to the labeled node with the same name. Let $|\mathcal{L}|$ be a total number of labeled nodes in $\mathcal{L}$. Then, the value of n is $\lceil \log_2 |\mathcal{L}| \rceil$. For instance, we need only 4-bit binary code to represent 16 different node labels. Secondly, it is a need to concatenate all n-bit binary codes on the same path from the root to each leaf node in a tree. We call the concatenated n-bit binary codes for each path *bit sequence*, abbreviated *bs*. Consequently, a set of trees $\mathcal{D}$ is transformed to a large set of bit sequences LBS.

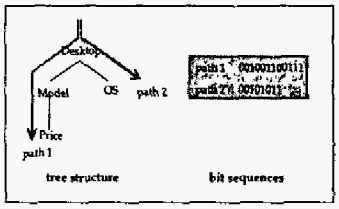


Fig. 2. A Tree Transformed to Bit Sequences

Example 2: Fig. 2 illustrates how an ordinary tree structure is represented by bit sequences. The binary code 0010 in bit sequences corresponds to the `Desktop` node label.

2) KTpSet Structure: One of features in EMaxS is the use of a new hierarchical dataset structure named *KTpSet (Key/Tlist pairs Set)* to find every frequent sets. KTpSet is kind of an array structure consisting of *key/tlist* pairs.

Definition 5 (Key): Given a set of bit sequences, let b_d is a set of unique n-bit binary codes placing at depth[1] d in every tree in $\mathcal{D}$. We assume that depth of root node is 0. We name each element of b_d *key*.

Definition 6 (Treeid): Given a set of bit sequences, each bit sequence *bs* is identified by its two subscripts. First one is an unique id of a tree and second one is a id of a sequence in a tree. We call the id of a tree *treeid*. Typically, treeid is numbered sequentially.

Definition 7 (Tlist): Let a key k be at depth d. The treeids of which bit sequences place k at the depth d consist of a list. We name the list *tlist*. A size of tlist is a number of treeids in the tlist.

Since all input trees are represented as a set of bit sequences, it is required to handle both n-bit binary codes and trees. Hence, we introduce a special data structure KTpSet to facilitate extracting maximal frequent subtrees.

Definition 8 (KTpSet): A *KTpSet* is defined as a set of pairs (k_d, t_{id}) where k_d is a key in K_d and t_{id} is a list of treeids to which k_d belongs. KTpSet typically indicates primary KTpSet denoted as $[\mathcal{P}]^d$.

3) KTpSet Classification: Every pair in $[\mathcal{P}]^d$ is classified according to its frequency.

Definition 9 (Frequent Code): Given some minimum support σ, let (k, t_k) be one of pairs in $[\mathcal{P}]^d$. The k is a *frequent code* if and only if the size of t_k is equal to or

[1] Virtually, *depth* indicates a number of edges in a path from a root to each node in a tree. Here, an n-bit binary code presents a single node, thus the depth is a number of n-bit binary code from the most significant bit to every n-bit binary code on the same sequence.

greater than $\sigma \times |\mathcal{D}|$.

Definition 10 (Frequent KT and Candidate KT): The pair whose key is a frequent code becomes an element of a *frequent KT*. Otherwise it is in *candidate KT*. We denote frequent KT and candidate KT by $[F]^d$ and $[C]^d$, respectively.

Example 3: Fig. 3 shows a structure of a KTpSet. As shown in the figure, three pairs exist in this KTpSet at some depth d.

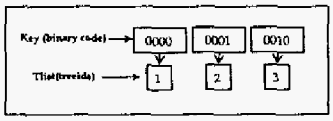


Fig. 3. KTpSet Structure

4) KTpSet Refinement: The initial set of frequent KT is similar to the set of frequent 1-edge subtrees (with one node). It is required to consider the characteristic of tree structure. This consideration stems from the fact that same labels can be placed several times throughout a XML tree. Previous researches typically joined k-edge frequent subtrees to generate $k+1$-edge candidate subtrees and pruned those candidates to get $k+1$-edge frequent subtrees. This routine is repeated until there is no frequent subtree any more. In EMaxS, we also generate candidate sets, however, it is accomplished without any join operation. To this end, we need to process more detailed operations called *depth–specific operation* according to each depth of both frequent and candidate KTs.

Example 4: The feature in Fig. 4 depicts the final sets of both frequent and candidate KTs after several rounds of depth-specific operations. Note that the given minimum support σ is $\frac{2}{3}$.

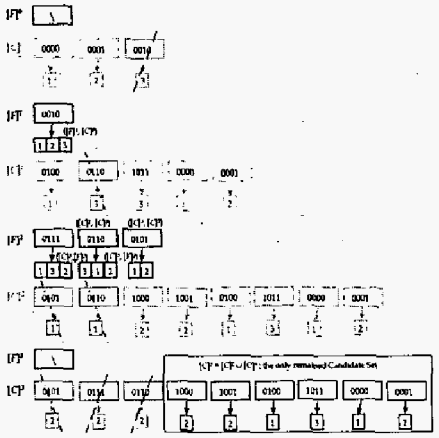


Fig. 4. Final Frequent KT

B. Maximal Frequent Subtree Construction

The final set of frequent KTs contain all of keys which are frequent codes. However, not every frequent KT has pairs, and not every n-bit key has relationship each other in original tree structures. For example, frequent KT $[F]^0$ and $[F]^3$ in Fig. 4 do not have any pairs. To facilitate constructing maximal tree patterns, we don't consider those empty frequent KT because an empty frequent KT implies that any n-bit binary code (in other word, any label) can be placed.

For maximal tree pattern generation with frequent KT, we need to find the firstly appearing nonempty frequent KT. The keys in the first nonempty frequent KT are eligible to become root nodes of maximal frequent subtrees. We incrementally expand those subtrees by the procedures described below; From the second nonempty frequent KT it is required to make edges between current keys and previously decided nodes. If a current key in $[F]^d$ shares enough treeids with any formerly made nodes to satisfy a minimum support, the key becomes a child node of those nodes. Otherwise, the key is eligible to be a root node of a new maximal frequent subtree if it cannot find any previous node sharing treeids as many as minimum support. Note that the subtrees having only one node are eliminated from the final set of maximal frequent subtrees since they are not considered as trees.

Example 5: Consider Fig. 5 showing the maximal $\frac{2}{3}$-frequent subtree built from the result in Fig. 4. The first met nonempty frequent KT has one element, thus the initial number of root nodes is one. In addition, the maximal depth of a newly constructed subtree cannot exceed 1 since only two frequent KT are nonempty. After the constructions of tree structures is completed, the original string labels are restored.

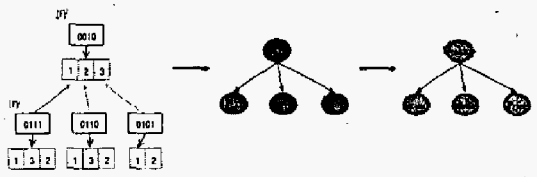


Fig. 5. Maximal $\frac{2}{3}$−frequent Subtree

V. Evaluation

To address the question of how feasible it is in practical to deploy EMaxS as a mining strategy, we carried out a couple of experiments. In this section, we describe our implementation, the test set used, and the results.

1) Experimental Setting: Our experiments were run on a AMD Athlon 64 3000+ 2.0 GHz, with 1GB RAM running Windows XP. The algorithm was implemented in Java.

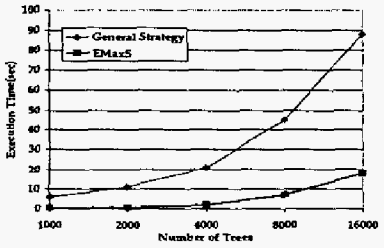
Two synthetic data set, S15 and S5, were tested. These data sets were generated by using the method; tree generator, in [8]. The generator mimics three target frequent trees reflecting predefined parameters. The parameters used in the tree generation include number of labels $N = 100$, the parameter of probability $\rho = 0.2$ including a target frequent tree at

each node, the maximum number of perturbation $\delta = 25$, the maximum fanout of a node in a tree $f = 5$, and the maximum distance $d = 3$, the minimum support σ ($\sigma = 0.15$ for S15, 0.05 for S5).

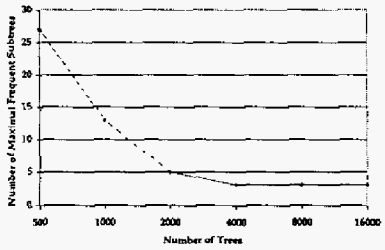
We set EMaxS against the most generic strategy. The basic idea behind the strategy is to identify frequent subtrees through a repeated process of enumerating and pruning candidate subtrees.

2) Results: The execution time for data set S15 with varying number of input trees is shown in Fig. 6(a). It can be seen that the two graphs have big difference in execution time. This is because the generation of thousands of string node objects based on join operations produces an exponential search space. Thus the overhead for checking candidate sets becomes a major bottleneck in the general strategy. This problem is improved dramatically due to not only binary coding representation but also no join operations.

The number of maximal frequent subtrees in Fig. 6(b) decreases regularly until numbers of input trees reach 4000. The total number of maximal frequent subtrees converges to the number of target frequent trees. Since we have considered fixed number of labels and the probability ρ of including three target frequent tree is set 0.2 when trees are generated, the possibility of placing same labels becomes high and all frequent patterns happen within the target frequent trees extensions.



(a) Scalability : Data set S15



(b) Output : Data set S5

Fig. 6. Experimental Results

VI. Conclusion

In a Web service environment, each maximal frequent subtrees plays an important role as clustering criteria for XML routers and as a symbolic characterization of each back-end server. This paper has outlined a new approach, called EMaxS, for discovering all maximal frequent subtrees of Web XML documents stored in web server's database. We suggest two unique techniques in the proposed algorithm. One is binary coding representation of all XML trees and the other is a special data structure KTpSet consisting of a set of key/tlist pairs.

We need further research to adapt the current proposed methodology to both synonym and polysemy of each element of XML trees, for use in XML mining operations, namely, clustering or classification in XML routers. We are currently working on more specific implementation of EMaxS.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases", Proceedings of the 12th International Conference on Very Large Databases, 1994, pp487–499.

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, "Efficient substructure discovery from large semi-structured data", Proceedings of 2nd SIAM International Conference on Data Mining (ICDM'02), 2002.

[3] P. Felber, C.-Y. Chan, M. Garofalakis, and R. Rastogi, "Scalabel Filtering of XML Data for Web Services", IEEE Internet Computing, 7 (1), 2003, pp. 49–57.

[4] M. Fernandez, and D. Suciu, "Optimizing regular path expressions using graph schemas", Proceedings of 14th International Conference on Data Engineering (ICDE'98), 1998, pp.14–23.

[5] A. Inokuchi, T. Washio, and H. Motoda, "An Apriori-based algorithm for mining frequent substructures from graph data", Proceedings of 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'00), 2000, pp. 13–23.

[6] M. Kuramochi, and G. Karypis, "Frequent subgraph discovery", Proceedings of IEEE International Conference on Data Mining (ICDM'01), 2001, pp.313–320.

[7] T. Miyahara, T. Suzuki, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda, "Discovery of frequent tag tree patterns in semistructured web documents", 6th Pacific-Asia Conference of Advances in Knowledge Discovery and Data Mining (PAKDD'02), 2002 pp.341–355.

[8] A. Termier, M-C. Rousset, and M. Sebag, "TreeFinder : a First step towards XML data mining", Proceedings of IEEE International Conference on Data Mining (ICDM'02), 2002, pp. 450–457.

[9] K. Wang, and H. Liu, 1998. "Schema discovery for semistructured data", Proceedings of 3rd International Conference on Knowledge Discovery and Data Mining (KDD'97), 1997, pp. 271–274.

[10] K. Wang, and H. Liu, "Discovering structural association of semistructured data", IEEE Transactions on Knowledge and Data Engineering (TKDE'00), vol.12, no.3, 2000, pp.353–371.

[11] M. J. Zaki, "Efficiently mining frequent trees in a forest", Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD'02), 2002, pp. 71–80.

[12] The World Wide Web Consortium (W3C). Simple Object Access Protocol http://www.w3.org/TR/soap/, 2003.

[13] The World Wide Web Consortium (W3C). Web Service Description Language (WSDL) 2.0 Part 1, http://www.w3.org/TR//2004/WD-wsdl20-20040803/, 2004.

[14] The World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation, http://www.w3.org/TR/2004/REC-xml-20040204/, 2004.