Extraction of Interesting Financial Information from Heterogeneous XML-Based Data*

Juryon Paik, Young Ik Eom, and Ung Mo Kim

Department of Computer Engineering, Sungkyunkwan University,
300 Chunchun-dong, Jangan-gu, Suwon,
Gyeonggi-do 440-746, Republic of Korea
quasa277@gmail.com, {yieom, umkim}@ece.skku.ac.kr

Abstract. XML is going to be the main language for exchanging financial information between businesses over the Internet. As more and more banks and financial institutions move to electronic information exchange and reporting, the financial world is in a flood of information. With the sheer amount of financial information stored, presented and exchanged using XML-based standards, the ability to extract *interesting* knowledge from the data sources to better understand customer buying/selling behaviors and upward/downward trends in the stock market becomes increasingly important and desirable. Hence, there have been growing demands for efficient methods of discovering valuable information from a large collection of XML-based data. One of the most popular approaches to find the useful information is to mine frequently occurring tree patterns. In this paper, we propose a novel algorithm, `FIXiT`,for efficiently extracting maximal frequent subtrees from a set of XML-based documents. The main contributions of our algorithm are that: (1) it classifies the available financial XML standards such as FIXML, FpML, XBRL, and so forth with respect to their specifications, and (2) there is no need to perform tree join operations during the phase of generating maximal frequent subtrees.

1 Introduction

XML, a meta-level data language with great flexibility and power, has emerged as the format of choice for data exchange across many industries and application domains. In particular, the financial world has long been a large and demanding user of information technology and there are a number of areas in which the finance community is looking to XML to improve business. Most of the emerging standards in financial data exchange rely on XML as the underlying structural language. Within companies, XML is being used to (1) integrate legacy systems, (2) disseminate news and information, and (3) make inroads for general financial

* This work was supported in part by the Ubiquitous Autonomic Computing and Network Project, 21st Century Frontier R&D Program and by the university IT Research Center project (ITRC), funded by the Korean Ministry of Information and Communication.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part IV, LNCS 3994, pp. 356–363, 2006.

transactions. XML establishes all these issues easier because, along with web services, it can take the data and build it easily accessible and flexible. The use of the web services to make it easier for systems to exchange and interrogate data without human intervention has generated the bulk of XML-based financial data.

With the rapidly increasing volume of the data, it becomes a new challenge to find useful information from a set of XML-based trees. In order to make the information valuable, it is important to extract frequent subtrees occurring as common trees embedded in a large collection of XML-based trees. However, as observed in previous studies [3, 4], because of the combinatorial explosion, the number of frequent subtrees usually grows exponentially with the tree size. Therefore, mining all frequent subtrees becomes infeasible for large tree sizes. In this paper, we present a novel algorithm, `FIXiT`, and a specially devised data structure for efficiently finding maximal frequent subtrees occurring mainly in a set of XML-based data. The proposed algorithm not only reduces significantly the number of rounds for tree pruning, but also simplifies greatly each round by avoiding time-consuming tree join operations. Toward this goal, our algorithm represents each node label of a XML-based tree as a binary code, stores them in specially devised data structures, and finds all maximal frequent tree patterns by expanding frequent sets incrementally.

The rest of this paper is organized as follows. We begin by reviewing some related works in Section 2. We continue in Section 3 with a description of some notions and definitions used throughout the paper. Then, we present our new algorithm FIXiT in Section 4. Finally, in Section 5 we sum up the main contributions made in this paper and discuss some of our future works.

2 Related Works

The various works for mining frequent subtrees are described in [2, 10, 11, 12]. Wang and Liu [11] considered mining of paths in ordered trees by using Apriori [1] technique. They propose the mining of wider class of substructures which are subtrees called schemas. Asai et al. [2] proposed FREQT for mining labeled ordered trees. FREQT uses rightmost expansion notion to generate candidate trees by attaching new nodes to the rightmost edge of a tree. Zaki [12] proposes two algorithms, TreeMiner and PatternMatcher, for mining embedded subtrees from ordered labeled trees. PatternMatcher is a level-wise algorithm similar to Apriori for mining association rules. TreeMiner performs a depth-first search for frequent subtrees and uses the scope list for fast support counting. Termier et al. [10] developed TreeFinder which uses a combination of relational descriptions for labeled trees and θ-subsumption notion to extract frequent subtrees.

The common problems of the previous approaches are identified as follows; (1) the number of frequent subtrees usually grows exponentially with the size of frequent subtrees, and therefore, mining all frequent subtrees becomes infeasible for large tree sizes. (2) The previous approaches represent each node of a XML tree as a labeled character string. This causes increasing the number of tree

pruning operations greatly, thus generating large number of candidate sets during the mining phase. Furthermore, each tree pruning round during generating candidate sets requires to perform expensive join operations. Therefore, as the number of XML documents increases, the efficiency for extracting frequent subtrees deteriorates rapidly since both the cost of join operations and the number of pruning rounds add up.

3 Preliminaries

In this section, we briefly review some notions of tree model and describe the basic concepts of mining for XML-based data.

Definition 1 (Labeled Tree). *A **labeled tree** is a tree where each node of the tree is associated with a label.*

Every XML-based data is represented by a labeled tree. For simplicity, in the remaining sections, unless otherwise specified, all trees are labeled. In addition, because edge labels can be subsumed without loss of generality by the labels of corresponding nodes, we ignore all edge labels in this paper.

Definition 2 (Subtree). *Let $T = (N, E)$ be a labeled tree where N is a set of labeled nodes and E is a set of edges. We say that a tree $S = (N_S, E_S)$ is a **subtree** of T, denoted as $S \preceq T$, iff $N_S \subseteq N$ and for all edges $(u, v) \in E_S$, u is an ancestor of v in T.*

Intuitively, as a subtree defined in this paper, S must not break the ancestor-descendant relationship among the nodes of T.

Let $\mathbb{D} = \{T_1, T_2, \ldots, T_i\}$ be a set of trees and $|\mathbb{D}|$ be the number of trees in $\mathbb{D}$.

Definition 3 (Support). *Given a set of trees $\mathbb{D}$ and a tree S, the frequency of S with respect to $\mathbb{D}$, $freq_{\mathbb{D}}(S)$, is defined as $\Sigma_{T_i \in \mathbb{D}} freq_{T_i}(S)$ where $freq_{T_i}(S)$ is 1 if S is a subtree of T_i and 0 otherwise. The **support** of S w.r.t $\mathbb{D}$, $sup_{\mathbb{D}}(S)$, is the fraction of the trees in $\mathbb{D}$ that have S as a subtree. That is, $sup_{\mathbb{D}}(S) = \frac{freq_{\mathbb{D}}(S)}{|\mathbb{D}|}$.*

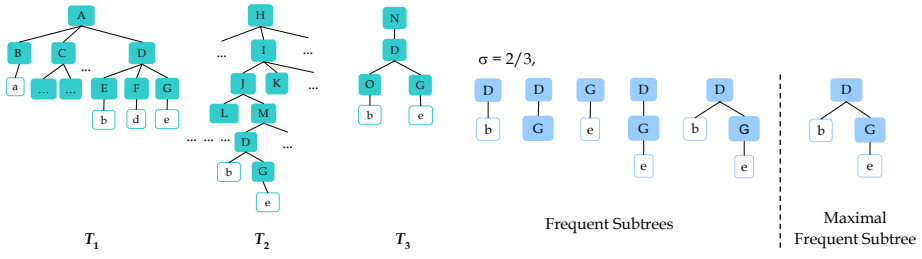
A subtree is called *frequent* if its support is greater than or equal to a minimum value of support specified by a user. This user specified minimum value of support is often called the *minimum support* (*minsup*).

The number of all frequent subtrees can grow exponentially with an increasing number of trees in $\mathbb{D}$, and therefore mining all frequent subtrees becomes infeasible for a large number of trees.

Definition 4 (Maximal Frequent Subtree). *Given some minimum support σ, a subtree S is called **maximal frequent** w.r.t $\mathbb{D}$ iff:*

i) the support of S is not less than σ, i.e., $sup_{\mathbb{D}}(S) \geq \sigma$.
ii) there exists no any other σ-frequent subtree S' w.r.t. $\mathbb{D}$ such that S is a subtree of S'.

Informally, a *maximal frequent subtree* is a frequent subtree none of whose proper supertrees are frequent. Usually, the number of maximal frequent subtrees is much smaller than the number of frequent subtrees, and we can obtain all frequent subtrees from the set of maximal frequent subtrees.



(a) Input: A set of XML-based trees $\mathbb{D}$ (b) Output: Maximal $\frac{2}{3}$-frequent subtree

Fig. 1. Maximal frequent subtrees of XML-based dataset

Example 1. An example of a set of XML-based trees $\mathbb{D}$ with various financial tags – we assume the each alphabet represents a unique financial vocabulary – is shown in Fig. 1(a). At a glance contents of three financial documents are different from each other and it seems that there is no similarity among them. However, when a minimum support value is given as $\frac{2}{3}$, the interesting hidden information is discovered, as illustrated in Fig. 1(b). With a sufficient reliability more than 60%, we can get to know the commonly-occurring financial information. Also with the same reliability, we find the implicit relations between financial tags; tag D is obtained always together with tag G.

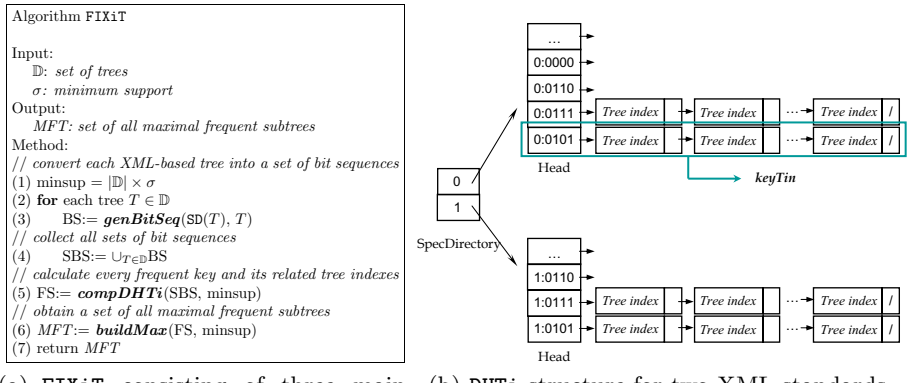
4 Overview of FIXiT

In this section, we describe main features of FIXiT (implicit Financial Information extraction by maXimal subTrees) algorithm which extracts a set of maximal frequent subtrees from a collection of XML-based financial data. The FIXiT builds on the mining algorithm EXiT-B presented in the recent work of Paik et al. [8, 9] and extends the EXiT-B algorithm: (1) to classify available financial XML standards with respect to their specifications, (2) to complement the weakness of the PairSet structure, and (3) even to mine conceptual semantics hidden in the XML data for future use. For a general overview of the algorithm EXiT-B and its PairSet structure, we refer the reader to [8, 9].

The purpose of the algorithm FIXiT is to efficiently find maximal frequent subtrees from a given set of XML-based data. Fig. 2 shows the outline of algorithm FIXiT and a specially devised data structure for it. As stated in Fig. 2(a), FIXiT consists of three functions: *genBitSeq, compDHTi*, and *buildMax*. The *genBitSeq* function classifies every XML-based document according to *SpecDirectoty*, and encodes each tree as a set of bit sequences by assigning an n-bit

binary code to each node label and concatenating the codes for all nodes on the same path. It takes as inputs a set of XML-based trees, and returns a set of bit sequences. The function *compDHTi* creates and maintains a specially devised data structure called, `DHTi` (`D`irectory–`H`eads–`T`ree `i`ndexes), in order to classify several XML specifications for the financial data, avoid join operations entirely during the phase of generating maximal frequent subtrees, and reduce the number of candidate subtrees from which frequent subtrees are derived. It uses as inputs the set of bit sequences which were originally XML-based trees and minimum support which is specified by a user, and produces frequent n-bit codes to be stored in the special structure DHTi. The *buildMax* extracts maximal frequent subtrees incrementally based on the n-bit binary codes stored in the DHTi produced by the function *compDHTi*. We omit the pseudo codes of three functions due to lack of space.

There are unique features that distinguish FIXiT to other XML mining algorithms: bit sequences representation of XML trees and DHTi data structure for storing each binary code along with its tree indexes. We look a little more deeply into those characteristics in the following subsections.



(a) `FIXiT` consisting of three main functions (b) `DHTi` structure for two XML standards

Fig. 2. Algorithm FIXiT and its data structure

4.1 Representation of Bit Sequences from a XML-Based Tree

Typical methods of representing a tree are an adjacency list [5], adjacency matrix [6], or character string [2, 7, 12]. 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 L be a set of labeled nodes in a set of trees $\mathbb{D}$. A function *genBitSeq* works as follows; First, it assigns an unique n-bit binary code randomly to each labeled node. Note that it must assign the same n-bit code to the nodes labels with the same name. Let $|L|$ be a total number of labeled nodes in L. Then, the value of n is $\lceil \log_2 |L| \rceil$. Second, it concatenates sequentially all the n-bit binary

codes on the same path. We call the concatenated n-bit binary codes for each path by a *bit sequence* (bs). Referring Fig. 2(a) again, BS denotes a set of bit sequences derived from a single tree in $\mathbb{D}$. Similarly, SBS denotes a collection of BSs derived from $\mathbb{D}$.

4.2 Generation of DHTi

Definition 5 (SpecDirectory). *Given $\mathbb{D}$, a **SpecDirectory** is defined as a directory structure that contains each index of XML-based financial standards.*

Definition 6 (Head). *Given a SBS, a **Head**, H_d, is defined as a collection of n-bit binary codes assigned on the nodes at each depth d of every tree in $\mathbb{D}$. We assume that depth of root node is 0. We call each member in H_d by a **key**.*

At this point, note that there may exist some nodes labeled with the same names in $\mathbb{D}$. Thus, for each key, we need to correctly identify the list of trees to which the key belongs.

Definition 7 (keyTin). *A **keyTin** is defined as a single pair of (k_d, t_{id}) where k_d is a key in H_d and t_{id} is a doubly linked list of tree indexes to which k_d belongs. A $[KT]^d$ is a set of all keyTins for a depth d.*

According to some minimum support, a collection of initial $[KT]^d$ is classified into two sets.

Definition 8 (Frequent keyTin). *Given some minimum support σ and a keyTin (key, t_{id}), the key is called **frequent** if $|t_{id}| \geq \sigma \times |\mathbb{D}|$.*

Definition 9 (Frequent Set). *Given $[KT]^d$, every keyTin in $[KT]^d$ becomes a member of **frequent set** if its key is frequent. Otherwise, it belongs to **candidate set**. We denote frequent set and candidate set by $[F]^d$ and $[C]^d$, respectively.*

The initial frequent sets correspond to the frequent subtrees with only one node commonly occurring in $\mathbb{D}$. Thus, we need to further extend these frequent sets incrementally to find final maximal frequent subtrees. For this purpose, we adopt the operation `cross-reference` introduced in [8, 9]. We refer to the reader to the paper [8] for a detailed explanation of `cross-reference`.

4.3 Construction of Maximal Frequent Subtrees

To derive maximal frequent subtrees from the final frequent sets, we need to notice the following two facts: Firstly, some of the final frequent sets may be empty. An empty frequent set at depth d indicates that there does not exist any n-bit binary code satisfying the minimum support at depth d. Thus, we do not need to consider those empty frequent sets for constructing maximal frequent subtrees. Secondly, although each frequent set has a hierarchical structure, not every key in the frequent set has connected each other in tree structures. In other words, some n-bit binary codes in different frequent sets have edges between them and some have not. It is required to decide whether an edge exists between two keys being in different frequent sets. A minimum support is used to make an edge between them.

5 Conclusion

As a financial community looks for reducing costs and increasing the accessibility of markets through off-the-shelf technologies and multi-vendor standards, XML is playing a key role as the technology which suddenly made everyone want to take part in an industry consortium and work with their competitors. Due to the the generation of a huge amount of the XML-based financial data, the flood of information problem has been caused. In this paper, we have introduced the novel algorithm, FIXiT, for finding valuable information from a collection of financial data. To this end the FIXiT extracts a set of all maximal frequent subtrees from the set of heterogeneous XML-based trees. Unlike previous approaches, the FIXiT classifies each XML-based document with respect to its financial specification, represents each node of a XML-based tree as a binary code, stores them in specially devised structures, and finds all maximal frequent subtrees by expanding frequent sets incrementally. The most important beneficial effect of the FIXiT is that it not only reduces significantly the number of rounds for tree pruning, but also simplifies greatly each round by avoiding time consuming tree join operations. We are currently working to evaluate the performance and the scalability of our algorithm through extensive experiments based on both synthetic data and datasets from real applications. Some of experimental results will be presented in the workshop.

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. Proceedings of the 12th International Conference on Very Large Databases (1994) 487–499
2. Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., Arikawa, S.: Efficient substructure discovery from large semi-structured data. Proceedings of the 2nd SIAM International Conference on Data Mining (2002) 158–174
3. Chi, Y., Yang, Y., Muntz, R. R.: HybridTreeMiner: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms. The 16th International Conference on Scientific and Statistical Database Management (2004) 11–20
4. Chi, Y., Yang, Y., Muntz, R. R.: Canonical forms for labelled trees and their applications in frequent subtree mining. Knowledge and Information Systems 8(2) (2005) 203–234
5. Inokuchi, A., Washio, T., Motoda, H.: An Apriori-based algorithm for mining frequent substructures from graph data. Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (2000) 13–23
6. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. Proceedings of IEEE International Conference on Data Mining (2001) 313–320
7. Miyahara, T., Suzuki, T., Shoudai, T., Uchida, T., Takahashi, K., Ueda, H.: Discovery of frequent tag tree patterns in semistructured web documents. Proceedings of the 6th Pacific-Asia Conference of Advances in Knowledge Discovery and Data Mining (2002) 341–355
8. Paik, J., Shin, D. R., Kim, U. M.: EFoX: a Scalable Method for Extracting Frequent Subtrees. Proceedings of the 5th International Conference on Computational Science. Lecture Notes in Computer Science, Vol. 3516. Springer-Verlag, Berlin Heidelberg New York (2005) 813–817

9. Paik, J., Won, D., Fotouhi, F., Kim, U. M.: EXiT-B: A New Approch for Extracting Maximal Frequent Subtrees from XML Data. Proceedings of the 6th International Conference on Intelligent Data Engineering and Automated Learning. Lecture Notes in Computer Science, Vol. 3578. Springer-Verlag, Berlin Heidelberg New York (2005) 1–8
10. Termier, A., Rousset, M-C., Sebag, M.: TreeFinder: a First step towards XML data mining. Proceedings of IEEE International Conference on Data Mining (2002) 450–457
11. Wang, K., Liu, H.: Schema discovery for semistructured data. Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (1997) 271–274
12. Zaki, M. J.: Efficiently mining frequent trees in a forest. Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (2002) 71–80